Markt&Technik

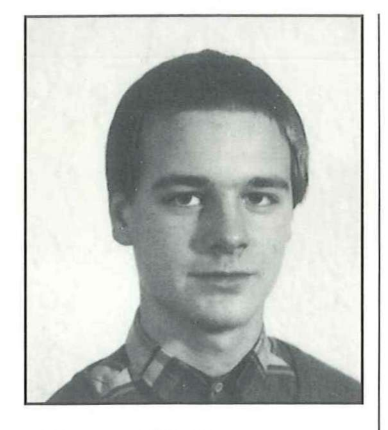
Florian Müller

Vom C64 zum C128

Tips & Tricks

Basic 7.0 für Fortgeschrittene · C 64-Programme umschreiben
Peeks & Pokes zum C 128 · Maschinensprache · Befehlserweiterung
Programmieren von Windows und komfortablen Auswahlmenüs





Vom C64 zum C128 Tips & Tricks

FLORIAN MÜLLER
geboren am 21. 1. 1970, besucht ein Münchner Gymnasium. Über seine Begeisterung für Videospiele und sein Interesse an der Mathematik fand er 1983 zum ZX Spectrum und kurze Zeit später zum C64, den er anfänglich vorwiegend als Spielemaschine einsetzte, aber bald in BASIC und Assembler programmierte.
Am meisten reizt ihn die Entwicklung einfacher Lösungswege für bestimmte Probleme. Dies zeigt sich an zahlreichen Artikeln rund um C16, C64 und C128, die zu den verschiedensten Themen Tips&Tricks vermitteln.
Ende 1985 erwarb er seinen jetzigen Lieblingscomputer C128; die Erfahrungen mit dem C128 im allgemeinen und dem Gerätewechsel im besonderen möchte er mit diesem Buch weitergeben.
Seit 1986 ist er außer für die Zeitschrift »64'er« auch für den Markt&Technik-Buchverlag als Autor und Lektor tätig.

Wenn Sie bereits mit dem C64 Erfahrung gesammelt haben, dann wird Ihnen das vorliegende Buch helfen, möglichst erfolgreich auf den C128 umzusatteln. Dabei erhalten Sie außer Grundlagenwissen auch viele Tips&Tricks und Hilfsprogramme, die die Arbeit mit dem C128 im C-64- und C-128-Modus erleichtern.
Bald ist Ihnen der C128 genauso vertraut wie der C64!

Aus dem Inhalt:
- Tastatur und Editor
- Die zwei Bildschirme
- Der deutsche Zeichensatz
- Von BASIC 2.0 zu BASIC 7.0
- ASCII-Codes von C64 und C128 im Vergleich
- Die neuen Befehle im Überblick
- Was nicht im Handbuch steht
- C-64-Programme umschreiben (BASIC und/oder Assembler)
- PEEKs&POKEs zum C128
- Anwendungen des BASIC 7.0
- Unterroutinen für professionelle BASIC-Programme
- Eingabefelder und Eingabemasken einfach programmiert
- Komfortable Menüs? - Kein Problem!
- Windows - Fenster zum Bedienungskomfort
- Befehlserweiterungen, Tips&Tricks zum BASIC 7.0 (alle wichtigen Befehle wie »OLD«, »FIND« und »MERGE«)
- Maschinensprache auf dem C128
- Der C-64-Modus und viele Hilfsprogramme dazu
- Einstieg in CP/M über einen Schnupperkurs
- Die Diskettenlaufwerke zum C128
- Kompatibilität

Nutzen Sie die unterschiedlichen Betriebssysteme des C128 für die Programmierung in BASIC und Assembler!

Alle Programme, darunter zahlreiche Utilities, sind auf der beigelegten Diskette (1541/1570/1571-Format) abgespeichert.

Hardware-Anforderung:
C128 oder C128D mit Floppy 1541, 1570 oder 1571

ISB N 3-89090-402-5



DM 49,-
sFr 45,10
öS 382,20





Florian Müller

Vom C64 zum C128 Tips & Tricks

- Basic 7.0 für Fortgeschrittene
- C64-Programme umschreiben
- Peeks & Pokes zum C128
- Maschinensprache
- Befehlserweiterung
- Programmieren von Windows und komfortablen Auswahlmenüs

Markt & Technik Verlag AG

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Müller, Florian:
Vom C64 zum C128 : Tips & Tricks ; Basic 7.0 für Fortgeschrittene ;
C-64-Programme umschreiben ; peeks & pokes zum C128 ; Maschinensprache ;
Befehlserweiterung ; Programmieren von windows und komfortablen Auswahlmenüs / Florian Müller. –
Haar bei München : Markt-und-Technik-Verlag, 1987. – & 1 Diskette
ISBN 3-89090-402-5

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
90 89 88 87

ISBN 3-89090-402-5

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Inhaltsverzeichnis

Vorwort

Als der Commodore 128 auf den Markt kam, erweckte eine seiner Eigenschaften das besondere Interesse der Computerwelt: der sensationelle C64-Modus.
Dessen Fähigkeit, die Unmengen vorhandener C64-Software zu verarbeiten, machte vielen, die sich von ihrem C64 nur ungern trennen wollten, aber über ihr Gerät hinausgewachsen waren, den Entschluß leicht, auf einen neuen Computer umzusteigen. Außerdem ist der C128-Modus eine Weiterentwicklung des C64 in den professionellen Bereich und und somit kein völliges Neuland.
Den Schritt des Umstieges (oder sollte man nicht »Aufstieg« sagen?) wird Ihnen das vorliegende Buch in jeder Hinsicht erleichtern; allerdings sollten Sie sich schon mit dem C128-Handbuch eingearbeitet haben und dieses Buch als Ergänzung verstehen. Daß dabei einiges wiederholt werden muß, was auch im Handbuch steht, läßt sich leider kaum vermeiden.
Im wesentlichen geht es uns hier aber darum, mit verhältnismäßig geringem Aufwand möglichst erfolgreich auf dem C128 zu arbeiten, indem wir unsere C64-Erfahrung effektiv einsetzen.
Ein Teil des Buches befaßt sich mit dem Erlernen des C128-Modus, wobei uns immer nur das beschäftigt, was gegenüber dem C64, den wir ja bereits kennen, anders ist, so daß Sie bald die zusätzlichen Befehle des mächtigen Basic 7.0 beherrschen, die neuen Tasten und Tastenbelegungen einsetzen und - bei vorhandenen Maschinensprachekenntnissen - in Assembler programmieren können, wie Sie es bislang auf dem C64 getan haben.
Andere Teile sind dem Umschreiben von C64-Programmen (Basic, Maschinensprache) gewidmet. Außerdem werden alle Eigenschaften, die den C64-Modus, also die Betriebsart, in welcher der C128 einen C64 simuliert, vom »Original«-C64 unterscheiden, behandelt. Der Einstieg in CP/M wird durch einen Schnupperkurs unterstützt und einiges zu den Diskettenlaufwerken, die für den C128 vorgesehen sind, verraten.
Die Kapitel sind nach Themenbereichen (Editor, Basic, Maschinensprache) gegliedert. Der überwiegende Teil des umfangreichen Basic-Abschnitts ist der Programmierung in Basic 7.0 für Fortgeschrittene gewidmet. Dabei werden nicht nur verständnisfördernde Anwendungen

zur Computergrafik und anderen Bereichen des Programmierens vorgestellt, sondern es wird auch vieles zur Programmierung von Ein-/Ausgabe (komfortable Menüs, Eingabefelder und Eingabemasken, Windows) in Kurzform vermittelt. Sie erhalten fertige Unterroutinen für die Praxis ebenso wie die Theorie dieser Programmiertechniken, um professionelle Programme schreiben zu können.
Zu jedem Thema finden Sie unzählige Tips und Tricks, vor allem in Form von wertvollen Hilfsprogrammen (Basic-Erweiterungen etc.), bei deren Einsatz die beigelegte Diskette sicher von Nutzen sein wird. Auf dieser befinden sich nicht nur die im Buch entwickelten Programme, sondern auch einige weitere, die aber selbstverständlich ausführlich im Text erläutert werden. So ist es möglich, Ihnen wesentlich mehr Software anzubieten, als dies in Form von Listings erreichbar wäre. Alle Programme, bei denen es sinnvoll ist, sind zwar auch abgedruckt, Sie können sich aber das mühselige und fehlerträchtige Abtippen ersparen, das einen Computer-Anwender oft zur Verzweiflung bringt.
Ich bin sicher, Sie werden die Programmdiskette bald nicht mehr missen wollen, denn wahrscheinlich erobert sie sich wegen der vielen nützlichen Programme einen festen Platz neben Ihrem C128.
An dieser Stelle möchte ich mich ganz besonders beim Markt & Technik-Buchverlag und der Redaktion der Zeitschrift 64'er für die große Unterstützung bedanken, die mir immer wieder in unterschiedlicher Form gewährt wurde.

Der Autor

1 Einführung

Damit Sie aus dem vorliegenden Buch den größtmöglichen Nutzen ziehen können, sollten Sie dieses Einführungskapitel lesen, das unnötige Probleme vermeiden hilft.

1.1 Der Umgang mit diesem Buch

Wir wollen hier wichtige Vereinbarungen treffen und auf eine Besonderheit, nämlich die beigelegte Programmdiskette, eingehen.

Die Programmdiskette

Wie Sie wissen, liegt diesem Buch eine Programmdiskette bei, die von den herkömmlichen Floppies (1541/70/71) gelesen werden kann.
Auf dieser sind
- alle im Buch abgedruckten Programme und
- weitere Programme, die ausführlich beschrieben werden,

abgespeichert. Insgesamt sind es 83 Programmfiles, von denen 57 als Listings abgedruckt sind. Das Listing eines Programms finden Sie dann vor, wenn dessen Funktionsweise wichtig ist. Der Filename jedes Listings wird mitangegeben. Dies soll Ihnen die Anwendung der Programmdiskette vereinfachen.
Sollte sich auf dieser als erster Eintrag im Directory ein File »LESER-INFO!« befinden, so laden Sie dieses bitte gleich; die Leser des 64er-Magazins kennen dies bereits von den Programmservice-Disketten zu dieser Zeitschrift.

Ein kleiner Tip: Machen Sie sich von der Programmdiskette eine Sicherheitskopie. Da die Programme nicht kopiergeschützt sind, können Sie jedes herkömmliche Kopierprogramm verwenden. Wenn Sie eine Floppy 1571 haben, können Sie das auf der 1571-Demodiskette befindliche Programm »DOS SHELL«, das im C1571-Handbuch in Kapitel 4 beschrieben wird, verwenden.

Schreibweise von Tastendrücken

Die Aufforderung, eine Taste zu drücken, wird folgendermaßen geschrieben: <TASTE>, also zum Beispiel <A> oder <5>.
Da Sie kein Anfänger sind, die bei <RETURN> die Buchstaben R,E,T,U,R und N einzeln eintippen würden, werden Steuertasten nicht anders als normale» (druckende) Tasten angegeben.
»CBM« bezeichnet dabei die Commodore-Taste (links unten auf der Tastatur).
Bei den Cursortasten werden folgende Bezeichnungen verwendet:

<CURSOR UP> = Cursor nach oben
<CURSOR DOWN> = Cursor nach unten
<CURSOR RIGHT> = Cursor nach rechts
<CURSOR LEFT> = Cursor nach links

Werden zwei Tasten gleichzeitig gedrückt, sind sie durch das Plus-Zeichen (»+«) verbunden.
Beispiel:

<SHIFT>+<RUN/STOP>

Zwei aufeinanderfolgende Tastendrücke werden dagegen durch ein Leerzeichen getrennt:

<X> <RETURN>

Welches Wissen wird vorausgesetzt?

Sie sollten bereits mit dem C64 gearbeitet und, da dieses Buch kein Handbuch-Remake ist, zumindest eine kurze Einführung aus dem Handuch entnommen haben.
Dennoch werde ich einiges wiederholen müssen, das auch im Handbuch zu finden ist, um Ihnen das Nachschlagen zu ersparen.
Eine Thematik, die für den Betrieb des C128 von größter Wichtigkeit ist, wird deshalb zu allererst behandelt: die verschiedenen Betriebsmodi (C128-Modus, C64-Modus, CP/M-Modus).
Wir werden uns in diesem Buch, sofern nicht am Anfang eines Kapitels oder schon in der Überschrift anders verlautet, mit dem C128-Modus beschäftigen. In diesen, der als einziger der drei Modi nur beim C128 vorhanden und also wirklich neu ist, gelangen Sie, wenn Sie Ihren C128 einschalten und abwarten, bis er seine Einschaltmeldung ausgegeben hat und der Cursor blinkt.
Die erste Änderung beim Einschalten ist schon, daß das angeschlossene Diskettenlaufwerk in Betrieb gesetzt wird, um eine CP/M-Diskette zu erkennen.

Halten Sie die <CBM>-Taste beim Einschalten gedrückt, wird der C64-Modus angewählt, d.h., Ihr C128 verhält sich wie der meistverkaufte Computer der Welt und kann dann auch C64-Programme ausführen. Näheres dann in Kapitel 5.
Der dritte und letzte Modus macht den C128 zu einer CP/M-Maschine, da dann das Betriebssystem CP/M (Control Program for Micro-Computers) auf ihm läuft. Dieses muß allerdings erst von Diskette eingelesen werden, weshalb man einfach vor dem Einschalten die mitgelieferte CP/M-Systemdiskette einlegt. Diese wird automatisch erkannt, es ist also nicht nötig, noch eine Taste gedrückt zu halten.
CP/M wird in Kapitel 6 dieses Buches angesprochen.

Der Reset-Schalter

Zum C64 muß ein Reset-Schalter separat erworben werden; der C128 verfügt über einen solchen von Natur aus, er befindet sich auf der rechten Gehäuseseite (siehe auch Kapitel 1 des C128-Handbuches).
Das Betätigen des Reset-Schalters führt zur Herstellung des Herstellung des Einschaltzustandes, ähnlich dem Aus- und Einschalten (allerdings geht durch einen Reset nur ein kleiner Teil des Speicherinhaltes verloren, während das Ausschalten den gesamten Speicher unwiederbringlich löscht).
Im Gegensatz zum C64-Reset besteht beim C128 die Möglichkeit, eine Moduswahl zu treffen:
- Reset ohne Tastendruck und CP/M-Diskette: C128-Modus
- Reset und <CBM>-Taste gedrückt gehalten: C64-Modus
- Reset und <STOP>-Taste gedrückt: Monitor

Wollen Sie aus dem CP/M-Modus über einen Reset in den C128-Modus zurück, müssen Sie vorher die CP/M-Systemdiskette aus dem Diskettenlaufwerk entfernen, da der C128 sonst erneut das Betriebssystem CP/M laden würde.
Wenn man bei einem Reset die <RUN/STOP>-Taste gedrückt hält, wird das Monitorprogramm angesprungen; sollte Ihnen dies versehentlich unterlaufen, kommen Sie durch <X> <RETURN> in den normalen Basic-Modus des C128.

1.2 Grundlagenwissen zum Editor

Auch wenn Sie sich sehr gut auskennen, sollten Sie diesen Abschnitt lesen, denn hier werden einige Begriffe erklärt, deren Auftauchen Ihnen an anderer Stelle (unnötige) Schwierigkeiten bereiten könnte.

Die Eingabemodi: Normal Mode, Quote Mode, Insert Mode

Solange man weder auf <SHIFT>+<2> (Anführungszeichen) noch auf <SHIFT>+ <INST/DEL> (Insert = Einfügen) drückt, befindet man sich im einfachsten Eingabemodus, dem

»Normal Mode« oder »Normalmodus«. Dieser hat die Eigenschaft, daß Steuertasten (zum Beispiel <CLR> = <SHIFT>+<CLR/HOME>, die Cursortasten oder die Farbsteuerung (=<CONTROL oder CBM>+<Zifferntaste>) sofort ausgeführt werden, was sich durch Löschen des Bildschirms, eine Cursorbewegung oder eine neue Cursorfarbe bemerkbar macht.
Damit man Steuerzeichen in Strings aufnehmen kann, gibt es aber noch den »Quote Mode«, den »Anführungszeichenmodus«. Diesen kennen Sie natürlich, denn er wird auch im C64/C128-Handbuch beschrieben. Im »Quote Mode« werden Steuerzeichen als reverse Buchstaben oder Grafikzeichen dargestellt.
In den »Quote Mode« gelangt man durch einmalige Eingabe eines Anführungszeichens; gibt man ein weiteres ein, befindet man sich wieder im »Normal Mode«, während erneutes < " > in den »Quote Mode« führt usw.
Schließlich gibt es noch den Insert Mode» (Einfügemodus), der in seiner Wirkung dem »Quote Mode« entspricht, wenn man davon absieht, daß im Insert Mode» zusätzlich noch die DEL-Taste als reverses Zeichen dargestellt wird.
Durch Betätigen von INST (= <SHIFT>+<INST/DEL>) kommt man in den »Insert Mode«, in dem man sich befindet, bis so viele Zeichen eingefügt wurden, wie oft <INST> gedrückt wurde. Dies wird ebenfalls im Handbuch erklärt (bei der Besprechung der <INST>-Taste), allerdings wird dort der Begriff »Insert Mode« nicht genannt, was wir eben nachgeholt haben.

2 Die Tastatur

Gegenüber dem C64 ist die C128-Tastatur in zweifacher Hinsicht verbessert worden. Erstens ist sie von besserer mechanischer Qualität, was vor allem für längere Schreibarbeiten (Textverarbeitung, Eingabe umfangreicher Programme usw.) von Vorteil ist, zweitens sind einige neue Tasten und Tastenfunktionen hinzugekommen, die neue Editorfunktionen bewirken. Wir wollen uns hier nur mit diesen neuen Tasten beschäftigen.
Auch wenn Sie sich mit der C128-Tastatur schon gut auskennen, sollten Sie dieses Kapitel lesen, denn hier wird neben einer kurzen Erklärung auch einiges über die Funktionsweise vermittelt. Außerdem finden Sie einige Tips und Tricks, die Ihnen den Umgang mit dem C128 erleichtern können.
Beachten Sie bitte, daß die nun folgenden Ausführungen nicht das Handbuch ersetzen sollen, sondern vielmehr als Ergänzung vorgesehen sind. Es ist also bestimmt nicht von Nachteil, wenn Sie vorher den (sehr kurzen) Abschnitt 4.1 des C128-Handbuches lesen oder schon gelesen haben.

2.1 Die neuen Tasten

In Ihrem C128-Handbuch finden Sie auf Seite 3-2 (Kapitel 3/Seite 2) eine Abbildung der Tastatur. Dabei sind die Tasten, die schon b4 vorhanden waren, blau gekennzeichnet. Im C64-Modus sind nur diese Tasten vorgesehen. Die anderen kann man teilweise aber ebenso einsetzen; so kann man beispielsweise mit der Taste <CAPS LOCK>, die beim 128D die Aufschrift <ASCII/DIN> trägt, auch im C64-Modus zwischen deutschem und amerikanischem Zeichensatz wählen, und mit Hilfe eines kleinen Maschinenprogramms, das in 5.2.5 vorgestellt wird

und sich auf der Programmdiskette befindet, können fast alle Tasten (auch <ESC>, <TAB> usw.) im C64-Modus abgefragt werden.
Besprechen wir nun die Tasten der Reihe nach, indem wir die grauen Tasten oben auf der Tastatur »von links nach rechts« behandeln.

ESC - vielseitig verwendbar

<ESC> für sich allein hat keine Funktion, sondern bewirkt nur, daß die nächste gedrückte Taste als sogenannte »ESC-Anweisung« interpretiert und ausgeführt wird. Dadurch ist es möglich, eine ganze Reihe Steuerfunktionen, die über die herkömmlichen Steuertasten wie <CLR/HOME> nicht erreichbar sind, aufzurufen. Die Möglichkeiten, die sich daraus ergeben, werden in 2.1.2 behandelt, da sie für eine kurze Abhandlung zu komplex sind.

TAB - die Tabulatortaste

Die <TAB>-Taste dient dazu, den Cursor mit einem einzigen Tastendruck mehrere Spalten weit zu bewegen; durch <TAB> wird er nämlich veranlaßt, bis zum nächsten »Tabulatorstop« zu springen. Ein solcher Tabulatorstop ist alle 8 Spalten vordefiniert.
Will man einen neuen Tabulatorstop setzen oder einen bereits eingestellten löschen, so fährt man mit den Cursortasten in die Spalte, in der der betreffende Tabulatorstop liegen soll, und drückt <SHIFT>+<TAB>. Dann gilt diese Spalte fortan als Tabulatorstop und kann folglich über TAB angesprungen werden.
Befand sich auf dieser Spalte vor der Betätigung von <SHIFT>+<TAB> bereits ein Tabulatorstop, so wird dieser gelöscht.
Um Mißverständnissen vorzubeugen: im Gegensatz zu manchen Textverarbeitungssystemen kann man beim C128 nur eine Spalte als Tabulatorstop definieren, nicht aber eine genaue Bildschirmposition, die aus Spalte und Zeile besteht. Folglich gilt der definierte Tabulatorstop in allen Bildschirmzeilen.
Mit <ESC><Y> (erst <ESC>, dann <Y> drücken !) kann man die Voreinstellung der Tabulatorstops wieder aufrufen (also alle 8 Spalten), mit <ESC><Z> ist es möglich, alle eingestellten Tabulatorstops zu löschen.
Soeben haben Sie übrigens einen Vorgeschmack bekommen, wie man die <ESC>-Taste einsetzt, die kurz vorher erwähnt wurde.

ALT - der Schlüssel zur eigenen Tastaturbelegung

Die <ALT>-Taste ist mit <SHIFT>,<CBM> und <CONTROL> zu vergleichen, denn man muß <ALT> immer in Verbindung mit einer anderen Taste drücken, um damit ein bestimmtes Zeichen zu erreichen. »ALT« ist die Abkürzung für »ALTernative key definition«, also »alternative Tastaturbelegung«. Leider muß man sich eine solche erst mit vielen POKE-Befehlen selbst erstellen, bis man in den Genuß kommt, seine erste selbstdefinierte Taste anzusteuern. Dabei darf es sich dann nicht um ein neu definiertes Zeichen handeln, sondern nur um ein Zeichen des aktuellen Zeichensatzes, so daß die Möglichkeiten der <ALT>-Taste doch recht eingeschränkt sind. Bislang ist mir noch kein Programm bekannt, das von <ALT> sinnvollen Gebrauch macht.

Falls Sie sich trotzdem dafür interessieren, steht alles Wichtige im Anhang J des C128-Handbuches.
Im Normalzustand bleibt <ALT> wirkungslos, da keine alternative Tastaturtabelle definiert ist.

CAPS LOCK - ASCII/DIN beim 128 D

Die nächste Taste dient zur Auswahl des Zeichensatzes und wird deshalb in 2.3 ausführlich behandelt; hier sei nur gesagt, daß man durch Verriegeln der <CAPS LOCK>- bzw. <ASCII/DIN>-Taste den deutschen DIN-, durch Entriegeln den amerikanischen ASCII-Zeichensatz anwählt. Wir werden in Zukunft nur von <CAPS LOCK> sprechen, die 128D-ler wissen dann schon, daß sie entsprechend <ASCII/DIN> drücken müssen.

HELP - Hilferuf an den Computer

Die Taste <HELP> können Sie drücken, wenn das Basic 7.0 einen Fehler meldet und Sie die fehlerhafte Zeile sehen wollen. Diese wird im 80-Zeichen-Modus ab der frühesten Stelle, an der der Fehler aufgetreten sein kann, unterstrichen, im 40-Zeichen-Modus, der über die Möglichkeit des Unterstreichens nicht verfügt, revers angezeigt.
Da die <HELP>-Taste im Prinzip eine zusätzliche Funktionstaste ist, wird diese noch einmal in 2.1.1 erklärt.

LINE FEED - Zeilenvorschub

<LINE FEED> entspricht in der Wirkung <CURSOR DOWN> und ist somit keine neue Funktion im eigentlichen Sinn. <LINE FEED> ist ein von Druckern übernommenes Steuerzeichen und hat auch den dort üblichen ASCII-Code 10.

40/80 DISPLAY - Auswahl des Darstellungsmodus

Die <40/80 DISPLAY>-Taste ist mit der Taste <CAPS LOCK> bzw. <ASCII/DIN> vergleichbar. Mit dieser kann man den Darstellungsmodus einstellen:

- <40/80> nicht eingerastet = 40-Zeichen-Modus
- <40/80> eingerastet = 80-Zeichen-Modus

Die <40/80 DISPLAY>-Stellung wird jedoch nur bei der Initialisierung vom Computer berücksichtigt, also in folgenden Situationen:
- Einschalten des Computers
- Reset, z.B. durch Drücken der Reset-Taste rechts am Gehäuse
- <RUN/STOP>+<RESTORE>

Näheres zu der Fähigkeit des C128, zwei voneinander unabhängige Bildschirme zu verwalten, erfahren Sie in 2.2.

NO SCROLL - die Unterbrechungstaste

Die <NO SCROLL>-Taste, der die Kombination <CONTROL>+<S> entspricht, ist eine Art Pausentaste. Bei Bildschirmausgaben prüft der Computer, ob <NO SCROLL> gedrückt wird; ist dies der Fall, wird der Computer in einen Wartezustand versetzt, der durch nochmaliges Betätigen von <NO SCROLL> oder einer anderen Taste beendet werden kann.
Wegen der Funktion der <NO SCROLL>-Taste sollten Sie die ESC-Befehle berücksichtigen, die in 2.1.2 behandelt werden.
Die Bezeichnung <NO SCROLL> ist etwas verwirrend, denn es könnte der Irrtum entstehen, daß diese Taste nur beim Abrollen des Bildschirms (»Scrolling«) von Bedeutung ist; Tatsache ist, daß jede Art von Bildschirmausgabe zu einer automatischen Prüfung der <NO SCROLL>-Taste führt, ob dabei der Bildschirm gescrollt werden muß oder nicht. Solange allerdings keine Bildschirmausgabe stattfindet, ist <NO SCROLL> außer Betrieb.
Sicher ist <NO SCROLL> recht nützlich, wenn man ein Programm listet oder einfach anhalten will. Es kann jedoch auch viele Gründe geben, warum man ein Programm dagegen absichern will. Hier hilft folgender POKE:

```
POKE 247,PEEK (247) OR 64
```

Aufgehoben wird dieser POKE mit

```
POKE 247,PEEK (247) AND 191
```

Dies kann auch leicht in Maschinensprache formuliert werden.

Damit haben wir die neuen Tasten oberhalb der Schreibmaschinentastatur besprochen, wenn man von den abgesetzten Cursortasten absieht, deren Funktion eindeutig ist: Sie entsprechen den gewohnten Cursortasten; als C64-Umsteiger wird man sich nur schwer an diese Cursortasten gewöhnen können.
Zusätzlich zu den neuen Tasten sind im C128-Modus einige Tastenfunktionen von althergebrachten Tasten hinzugekommen.

CONTROL - Zusatzfunktionen aller Art

Die Wirkung der <CONTROL>-Taste des C64 wird nun durch <CBM> hervorgerufen: Bei Scrolling (Abrollen) des Bildschirms wird bei gedrückter <CBM>-Taste eine Verzögerung der Bildschirmausgabe hervorgerufen, die vor allem bei der Programmanalyse mit dem LIST-Befehl hilfreich ist.
Der <CONTROL>-Taste, welche beim C64 nur mit der Abkürzung »CTRL« beschriftet ist, kommt jetzt besondere Bedeutung zu. Sie ermöglicht noch eine Reihe Funktionen, die durch gleichzeitiges Betätigen von <CONTROL> und einer anderen Taste erreicht werden.
Hierzu ist zu sagen, daß schon beim C64 die <CONTROL>-Taste mächtiger ist, als es im Handbuch steht.
Aufgrund des Verfahrens, wie Commodore-Computer mit Steuerzeichen umgehen, können viele der Steuerzeichen, die in Basic über »PRINT CHR$(X)« aufgerufen werden, auch mit <CONTROL> im Eingabemodus eingegeben werden. Im C64(-Modus) gilt Tabelle 2.1:

Tabelle 2.1: *CONTROL-Funktionen des C64*

Taste mit <CONTROL>	ASCII-Code	Funktion
E	5	weiß (=<CONTROL>+<2>)
H	8	<SHIFT>+<CBM> verbieten
I	9	<SHIFT>+<CBM> zulassen
M	13	= <RETURN>
N	14	Klein-/Großschrift ein
Q	17	= <CURSOR DOWN>
R	18	= <REVERS ON>
S	19	= <HOME>
T	20	= <DELETE>
eckige Klammer zu	29	= <CURSOR RIGHT>

Wenn Sie eine solche <CONTROL>-Anweisung im Quote Mode eingeben, wird sie - von <M> und <T> abgesehen - nicht ausgeführt, sondern Sie können das in der Tabelle ganz links angegebene Zeichen in Reversdarstellung sehen. So ist es möglich, auch Codes, für die früher CHR$(x)-Kommandos nötig waren, in Strings als Steuerzeichen aufzunehmen; allerdings leidet die Übersichtlichkeit eines Programms stark darunter, weshalb die Listings in diesem Buch von dieser Möglichkeit keinen Gebrauch machen.

Im C128-Modus zeigen noch weitere <CONTROL>-Kombinationen Wirkung, außerdem liegen die Ver- und Entriegelfunktionen für <SHIFT>+<CBM> auf anderen Tasten als beim C64 (Tabelle 2.2):

Zu diesen Tabellen ist zu sagen, daß es sicher nicht sinnvoll ist, ohnehin vorhandene Tasten wie <RETURN> oder <DEL> über <CONTROL>-Verrenkungen zu simulieren; manche Funktionen sind aber auf andere Weise nicht erreichbar. Zwei Beispiele:

- Wenn <SHIFT>+<CBM> von einem Programm aus verhindert wurde und man aber dennoch zwischen Groß-/Grafik-Schrift und Klein-/Groß-Schrift wählen möchte, kann man bei einer INPUT-Eingabe mit <CONTROL>+<L> diese Schutzmaßnahme des Programms aufheben. Voraussetzung Nummer eins ist dabei aber, daß das Programm den Basic-Befehl »INPUT« und keine spezielle Eingabe-Routine, die gegen «CONTROL»-Tricks abgesichert ist, verwendet.
- Mit <CONTROL>+<G> kann man bei der Programmierung von Soundeffekten auf einfache Weise einen bei einer Fehlermeldung aufgetretenen Dauerton abstellen; es ertönt kurz das Klingelzeichen, danach hat der Dauerton aufgehört.

Tabelle 2.2: *CONTROL-Funktionen des C128*

Taste mit <CONTROL>	ASCII-Code	Funktion
B (nur 80-Zeichen-Modus)	2	Unterstreichmodus ein
E	5	weiß (= <CONTROL>+<2>)
G	7	akustisch Klingelzeichen
I	9	= <TAB>
J	10	= <LINE FEED>
K	11	<SHIFT>+<CBM> verbieten
L	12	<SHIFT>+<CBM> zulassen
M	13	= <RETURN>
N	14	Klein-/Großschrift ein
O (nur 80-Zeichen-Modus)	15	alle Zeichen blinken
Q	17	= <CURSOR DOWN>
R	18	= <REVERS ON>
S (anders als beim C64 !)	-	= <NO SCROLL>
T	20	= <DELETE>
X	24	= <SHIFT>+<TAB>
eckige Klammer auf	27	= <ESC>
eckige Klammer zu	29	= <CURSOR RIGHT>

2.1.1 Die Funktionstastenbelegung

Im eigentlichen Sinne sind die Funktionstasten des C128 keine Neuerung, denn auch beim C64 sind sie vorhanden. Der Unterschied besteht in der Behandlung durch das Betriebssystem. Die C64-Funktionstasten sind mit den Codes 133-140 belegt; dies sind Steuerzeichen, die keine Wirkung hervorrufen und lediglich der programmgesteuerten Funktionstastenabfrage, wie sie beispielsweise in einem Menüprogramm verwendet werden kann, Nutzen bringt. Eine Eingabe im Direktmodus ist jedoch ohne Wirkung (außer im Quote Mode, wo ein reverses Zeichen entsteht).

Anders beim C128. Dort bewirkt das Betätigen einer Funktionstaste, daß der Computer mehrere Tastendrücke simuliert (unabhängig vom Eingabemodus), was eine deutliche Erleichterung ist. Dies kennen Sie sicher vom C64 auch, da für diesen eine riesige Anzahl von Programmen geschrieben wurde, die eine Belegung der Funktionstasten mit (sinnvollen) Texten ermöglichen. Folgende Funktionstastenbelegung (Tabelle 2.3) ist voreingestellt und nach dem Einschalten des C128 sofort verfügbar (allerdings nicht im C64- oder CP/M-Modus!):

Tabelle 2.3: *Funktionstastenbelegung im C128-Modus*

Funktionstaste	Text	Beschreibung der Wirkung
<F1>	GRAPHIC	Grafik-Modus einstellen
<F2>	DLOAD"	Programm von Disk laden
<F3>	DIRECTORY <RETURN>	Directory anzeigen
<F4>	SCNCLR <RETURN>	Bildschirm löschen
<F5>	DSAVE"	Programm auf Disk saven
<F6>	RUN <RETURN>	Programm starten
<F7>	LIST <RETURN>	Programm listen
<F8>	MONITOR <RETURN>	Monitorprogramm starten
<SHIFT>+<RUN/STOP>...	DLOAD"* <RETURN> und RUN <RETURN>	erstes Programm von Disk laden und gleich starten
<HELP>....................	HELP <RETURN>	Fehler im Programm orten

Individuelle Funktionstastenbelegung über KEY

Vielleicht wundern Sie sich über die beiden letzten Funktionstasten; daß diese in obiger Tabelle auch aufgeführt sind, hat aber durchaus einen Grund, denn vom Betriebssystem wird <SHIFT>+<RUN/STOP> als F9 und <HELP> als F10 behandelt. Der Basic-Befehl zum Anzeigen und Verändern der Funktionstastenbelegung greift jedoch nur auf die herkömmlichen Funktionstasten <F1>-<F8> zu. Er heißt KEY und ist mit dem gleichnamigen Befehl aus Simon's Basic für den C64 zu vergleichen.
Mit

```
KEY
```

wird die aktuelle Funktionstastenbelegung (F1-F8) angezeigt, wofür Simon's Basic den Befehl KEY0 hat.

Durch

```
KEY n,"Text"
```

legt man einen Text, der in Anführungszeichen stehen muß, auf die Funktionstaste »n«. Der Parameter »n« hat einen Wert von 1 bis 8.
Beispiel:

```
KEY 1,"CLR"
```

legt den Text »CLR« auf die Funktionstaste 1. Da die Funktionstastenbelegung jedoch nur eine simulierte Tastatureingabe des Textes, nicht aber die Ausführung durch den Basic-Interpreter veranlaßt, muß noch der Code für RETURN angehängt werden, der dann bei Auslösen von <F1> vortäuscht, daß <RETURN> gedrückt wird:

```
KEY 1,"CLR"+CHR$(13)
```

Außer CHR$(13) kann man auch andere Steuerzeichen mittels CHR$-Befehl verwenden:

<SHIFT>+<RETURN> = CHR$(141)
Anführungszeichen = CHR$(34)
<ESC> = CHR$(27)

LOAD und SAVE statt DLOAD und DSAVE

Wenn Sie eine Datasette verwenden und auf <F2> lieber den Text LOAD" als DLOAD" und auf <F5> lieber SAVE" als DSAVE" hätten, müssen Sie nicht die umständliche Neubelegung vornehmen. Es geht auch einfacher, indem Sie vor Betätigen von »F2« oder »F5« auf »ESC« drücken.
Warum das »D« von DLOAD und DSAVE durch vorhergehendes »ESC« verschluckt wird, können Sie sich nach Lektüre von 2.1.2 selbst erklären.

Funktionstasten auf CHR$-Codes zurücksetzen

Beim C64 sind die Funktionstasten <F1>-<F8> mit Steuercodes (133-140) belegt. Dadurch erleichtern sie zwar die Eingabe nicht, aber die Funktionstastenabfrage ist ohne Schwierigkeiten als Vergleich mit CHR$-Codes (IF A$=CHR$(135) ...) zu programmieren.
Das C128-Programm

```
10 GETKEY A$:PRINT A$:GOTO 10
```

würde nach Drücken von <F1> die Zeichen »G«,»R«,»A«,»P«,»H«,»I« und »C« nacheinander mit dem GETKEY-Befehl empfangen, aber nie ein Zeichen, das eindeutig Aufschluß darüber gibt, daß »F1« gedrückt wurde.
Folgende Programmzeile für den C64(-Modus) wartet auf <F1>:

```
10 GET A$:IF A$<>CHR$(133) THEN 10
```

Dies ist, da beim C64 der Code 133 die Taste <F1> repräsentiert, unproblematisch.
Unter Zuhilfenahme des »KEY«-Befehls können Sie diese Belegung mit den Codes 133-140 auch im C128-Modus bewirken, um dann die Funktionstasten genauso einfach wie mit dem C64 abfragen zu können:

```
KEY 1,CHR$(133):KEY 2,CHR$(137):KEY 3,CHR$(134):KEY 4,CHR$(138):
KEY 5,CHR$(135):KEY 6,CHR$(139):KEY 7,CHR$(136):KEY 8,CHR$(140)
```

Dann verhalten sich die Funktionstasten entsprechend dem uns gut bekannten C64, weshalb sich eine weitere Erklärung erübrigt.
Zwei kleine Tips dazu:
- Setzen Sie diese KEY-Befehle gleich in die erste Zeile eines Programms, das davon Gebrauch macht.
- Zur Funktionstastenabfrage empfiehlt sich ausnahmsweise der Einsatz des Quote Mode, da die Codes etwas unglücklich organisiert sind (während F1 den Code 133 hat, trägt F2 nicht

den Code 133+1 = 134, sondern den Wert 137 usw.; es gibt also keine einfache Additions- oder Subtraktionsformel, um aus dem CHR$-Code die Nummer der gedrückten Funktionstaste zu ermitteln.).

Abspeichern der Funktionstastenbelegung

Wenn man sich seine individuelle Funktionstastenbelegung zusammengestellt hat, empfiehlt es sich, diese auf Diskette abzuspeichern, damit man nicht nach jedem Einschalten des Computers eine »KEY«-Orgie eingeben muß. Für diejenigen, die sich für den Maschinensprache-Monitor nicht interessieren, ist es z.B. ratsam, <F8> mit einem für ihre Zwecke nützlicheren Kommando zu belegen, wofür der Befehl »PRINT DS$« zur Anzeige einer Floppy-Fehlermeldung in Frage kommt. Auch auf die Funktionstaste <F4> kann man gut verzichten, da diese nur mit dem Befehl SCNCLR belegt ist, den man meist durch <SHIFT>+<CLR/HOME> ersetzen kann.
So sichert man die aktuelle Funktionstastenbelegung auf Diskette:

```
BSAVE "F-BELEGUNG",ON B0,P4096 TO P4352
```

Der Filename »F-BELEGUNG« kann dabei durch jeden anderen zulässigen Text ersetzt werden.
Mit folgendem Befehl wird sie wieder in den Speicher geladen:

```
BLOAD "F-BELEGUNG",ON B0
```

<SHIFT>+<RUN/STOP> ausschalten

Wenn man beim Programmieren in Basic 7.0 versehentlich die Kombination <SHIFT>+<RUN/STOP> auslöst, wird das erste Programm von Diskette geladen und automatisch gestartet, was die gravierende Folge hat, daß das im Speicher befindliche Programm überschrieben wird und somit unwiederbringlich verloren ist.
Dieses Problem löst man am einfachsten durch Verzicht auf diese Belegung von <SHIFT>+<RUN/STOP>. Folgender Befehl, dessen Funktionsweise ich hier nicht ausbreiten will, schaltet die Kombination <SHIFT>+<RUN/STOP> ab:

```
BANK 15:SYS DEC ("C021"),,9,0
```

Da dieser SYS recht schwer zu merken ist, empfiehlt es sich, bei der Erstellung einer eigenen Funktionstastenbelegung nach folgendem Schema vorzugehen:
1) C128 anschalten, um Vorbelegung zu bewirken
2) SYS DEC ("C021"),,9,0
3) Mit KEY die gewünschte Belegung einstellen
4) BSAVE "F-BELEGUNG",ON B0,P4096 TO P4352

Später kann man diese, wie schon gesagt, mit

```
BLOAD "F-BELEGUNG", ON B0
```

wieder in den Speicher holen.

2.1.2 Die ESC-Taste

Wie wir in diesem Kapitel 2 bei der Besprechung der <CONTROL>-Taste gesehen haben, verfügt der C128-Editor über eine Vielzahl nützlicher Funktionen. Die vielfältig verwendbare <ESC>-Taste, die sich links oben auf der Tastatur befindet, erhöht die Anzahl der Möglichkeiten ganz beträchtlich.
Im Gegensatz zu <CONTROL> betätigt man <ESC> nie gleichzeitig mit einer anderen Taste, sondern man drückt zuerst <ESC> und dann eine Taste, die schließlich die gewünschte Funktion aufruft.
Der C128 interpretiert also nach Drücken von <ESC> den nächsten Tastendruck als Kommando; deshalb deshalb wird er nicht am Bildschirm angezeigt.
Damit Sie an einem einfachen Beispiel die Wirkung einer Escape-Sequenz» (»ESC« ist die Abkürzung für »Escape«, was wörtlich »entkommen, flüchten« bedeutet) sehen, versetzen Sie Ihren C128 bitte in den Einschaltzustand und betätigen Sie folgende Tasten:

<ESC><E>

Dann verwandelt sich der blinkende Cursor in einen Festcursor, wie Sie ihn vielleicht von manchen anderen Computern oder Ihrer Textverarbeitung her kennen.
Treffen wir hier eine kleine Vereinbarung: von jetzt an werden wir beispielsweise <ESC><E> als »ESC-E« schreiben, wobei natürlich der Bindestrich nicht eingegeben werden muß.
Die ESC-Sequenzen stehen in Tabelle 2.4 (in Klammern die Bedeutung der Abkürzungen; bei den Befehlen, die nur im 80-Zeichen-Modus wirken, steht ein »*«).
Anhand dieser Tabelle können Sie schon einmal die ESC-Sequenzen ausprobieren. Es folgen nun eingehendere Erklärungen zu einigen Befehlen, bei denen dies nötig ist.

Der automatische Einfügemodus

Vom C64 her kennen wir drei Eingabemodi: Normal Mode, Quote Mode und Insert Mode.
Beim C128 gibt es sogar noch einen vierten, nämlich den »Auto Insert Mode« (= automatischer Einfügemodus), der über ESC-A aktiviert wird. In diesem wird jedes eingegebene Zeichen an der Cursorposition eingefügt. Während bei <INST> nur ein einzelnes Zeichen eingefügt wird, ist der »Auto Insert Mode« immer aktiv, bis er über ESC-C beendet wird.
Ein weiterer wesentlicher Unterschied zu <INST>:
Im Gegensatz zum Insert Mode, in dem Steuertasten wie im Quote Mode behandelt werden, werden beim automatischen Einfügen alle Steuertasten direkt ausgeführt. Dies ist ein großer Vorteil, weil es so auch möglich ist, die Cursor- und andere Steuertasten zu verwenden.

Bildschirm-Scrolling verbieten/zulassen

Der C128 löst ein Bildschirm-Scrolling immer dann aus, wenn mehr Zeilen ausgegeben werden, als auf den Bildschirm passen. Das ist fast immer der Fall, wenn ein Programm gelistet wird.
Mit ESC-M kann man das Bildschirm-Scrolling abschalten; dann wird nach der Ausgabe der

Tabelle 2.4: *ESC-Sequenzen im Überblick*

ESC-A	(Auto Insert Mode = automatischer Einfügemodus)
ESC-B	(Set Bottom = Bildschirmuntergrenze setzen)
ESC-C	(Cancel Auto Insert Mode = automatischen Einfügemodus aus)
ESC-D	(Delete Line = Cursorzeile löschen)
ESC-E	(Gegenteil von ESC-F = Festcursor einschalten)
ESC-F	(Cursor Flash Mode = Cursor blinken lassen)
ESC-G	(in Analogie zu CONTROL-G: Klingelzeichen zulassen)
ESC-H	(Gegenteil von ESC-G, <CONTROL>+<G>-Signalton verbieten)
ESC-I	(Insert Line = Zeile an Cursorposition einfügen)
ESC-J	(Jump to Beginning of Line = an Zeilenanfang springen)
ESC-K	(Gegenteil von ESC-J, Cursor an Zeilenende setzen)
ESC-L	(Let Scrolling = Bildschirm-Scrolling zulassen)
ESC-M	(Gegenteil von ESC-L, Bildschirm-Scrolling verbieten)
ESC-N	* (Normal Screen = Gegenteil von ESC-R = Reversmodus aus)
ESC-O	(Off = Insert Mode, Quote Mode und Reversschrift aus)
ESC-P	(aktuelle Zeile bis zur Cursorspalte löschen)
ESC-Q	(aktuelle Zeile von Cursorspalte an löschen)
ESC-R	* (ganzen 80-Zeichen-Bildschirm invertieren)
ESC-S	* (Gegenteil von ESC-S, Block-Cursor einschalten)
ESC-T	(Set Top of Window = linke obere Window-Ecke setzen)
ESC-U	* (Underline Cursor = Unterstreich-Cursor ein)
ESC-V	(Bildschirm-Scrolling um eine Zeile nach oben)
ESC-W	(Bildschirm-Scrolling um eine Zeile nach unten)
ESC-X	(Umschalten zwischen 40- und 80-Zeichen-Modus)
ESC-Y	(Voreinstellung der Tabulatoren, also alle 8 Spalten)
ESC-Z	(Löschen aller Tabulatoren)
ESC-@	(Bildschirm ab Cursorposition löschen)
ESC-ESC	(entspricht ESC-O, ist aber leichter einzugeben)

letzten Bildschirmzeile wieder links oben, in der HOME-Position, die Bildschirmausgabe fortgesetzt.
Durch ESC-L ist es möglich, dieses Verbot wieder aufzuheben.
Es sei ausdrücklich darauf hingewiesen, daß mit ESC-M die <NO SCROLL>-Taste nicht beeinflußt wird. Wie man <NO SCROLL> abschaltet, steht am Anfang von 2.1.

ESC-B und ESC-T - Bildschirm verkleinern

Um den Bildschirm zu verkleinern, stehen die Sequenzen ESC-B und ESC-T zur Verfügung. ESC-T setzt die Cursorposition als linke obere Ecke, ESC-B als rechte untere Grenze. Dadurch können Sie ein beliebiges Bildschirmformat einstellen, auf das der Editor Rücksicht nimmt,

indem er z.B. Steuerzeichen wie <CLR> oder <HOME> auf das neue Format bezieht. Nach <HOME> <HOME> (Sie haben richtig gelesen, zweimal hintereinander muß <HOME> gedrückt werden), steht wieder der ganze Bildschirm (40 oder 80 Spalten und 25 Zeilen) zur Verfügung.
Mehr dazu finden Sie in Abschnitt 2.1.3, der die Window-Technik unter die Lupe nimmt.

ESC-X - Auswahl des 40- oder 80-Zeichen-Bildschirms

Mit ESC-X kommt man vom 40-Zeichen-Modus in die 80-Zeichen-Darstellung und umgekehrt. Dabei benötigt man jedoch entweder zwei Monitore (einen RGB- und einen Composite-Monitor) oder einen, der sowohl über RGB- als auch Composite-Darstellung verfügt. Dies wird ausführlich in 2.2 und auch in Kapitel 1 des C128-Handbuches erläutert.

ESC-@ im deutschen Zeichensatz

Um den Bildschirm ab der Cursorposition zu löschen, betätigt man im (amerikanischen) ASCII-Zeichensatz einfach ESC-@.
Der Klammeraffe (so nennt man das Zeichen »@«) ist zwar auf der (deutschen) DIN-Tastatur auch erreichbar, allerdings hat er dort einen anderen Code und wird von der Escape-Routine nicht erkannt. Deshalb muß man im DIN-Zeichensatz, den man durch Verriegeln der Taste <CAPS LOCK> (beim 128D: <ASCII/DIN>) einschaltet, statt dessen das Paragraphenzeichen eingeben, das man über <SHIFT>+<3> erreicht.

ESC-Sequenzen auf Funktionstasten

Bekanntlich kann man mit dem KEY-Befehl in die Funktionstastenbelegungen auch Steuertasten (<RETURN> usw.) integrieren. Interessant ist auch der Einsatz von ESC-Sequenzen. Nach

```
KEY 7,CHR$(27)+"T"+"LIST"
```

wird bei Betätigen von <F7> zuerst die aktuelle Cursorposition als linke obere Bildschirmecke festgelegt, was den Vorteil hat, daß bei einem Scrolling (was bei LIST fast immer auftritt) der Bildschirm bis zur Cursorposition geschützt ist (siehe auch 2.1.3).

E-Sequenzen über PRINT ausgeben

Mit

```
PRINT CHR$(27)"<ESC-Kommando>";
```

wird eine ESC-Anweisung ausgeführt.
Da manche Basic-Befehle eine andere Syntax als PRINT haben, ist man oft gezwungen, die CHR$(27)-Anweisung (27 = Code für ESC) und das ESC-Kommando mit dem Operator + zu verknüpfen, also z.B.:

```
LET A$=CHR$(27)+"D"
```

ESC im Quote Mode

Wahrscheinlich haben Sie schon bemerkt, daß ESC (wie RETURN und DELETE) vom Eingabemodus unabhängig ist. Dies hat den Vorteil, daß man z.B. mit ESC-ESC (= ESC-O) den Quote Mode nach Belieben abschalten kann, aber den kleinen Nachteil, daß eine Aufnahme von ESC in einen String als reverses Zeichen mittels Quote Mode, wie es etwa bei <CLR> oder <HOME> funktioniert, unmöglich ist.
Nach folgendem POKE-Befehl wird ESC im Quote Mode als reverses Zeichen dargestellt:

```
POKE 820,189
```

Mit folgendem Befehl läßt sich dies wieder rückgängig machen:

```
POKE 820,185
```

2.1.3 Die Window-Technik

Während man beim C64 und im C64-Modus des C128 immer nur den ganzen Bildschirm editieren kann und es nicht möglich ist, einen Teilbereich unabhängig vom übrigen Bildschirm zu behandeln, wurde in das Betriebssystem des C128 eine sogenannte »Window-Technik« eingebaut.
»Window« ist das englische Wort für »Fenster« und soll darauf hinweisen, daß nicht der ganze Bildschirm, sondern nur ein Ausschnitt (Fenster) als eigenständiger Bildschirm behandelt wird.
Das hat den großen Vorteil, daß sich dann alle Bildschirm-Operationen (Eingaben und Ausgaben jeder Art) auf dieses Window beziehen und somit der Restbereich des Bildschirms gegen Überschreiben geschützt ist.
Hierbei ist noch zu erwähnen, daß die Window-Technik nicht vom Video-Chip gesteuert wird (wie man irrtümlich meinen könnte), sondern vollständig dem Betriebssystem unterliegt. Deshalb kann man auch sowohl im 40- als auch im 80-Zeichen-Modus ein Bildschirmfenster definieren, obwohl im 40-Zeichen-Modus ein anderer Video-Chip (der VIC) als im 80-Zeichen-Modus (der VDC) zuständig ist.
Es ist zwar möglich, auf jedem der beiden Bildschirme ein Window zu definieren; allerdings ist das Betriebssystem nicht in der Lage, zwei Fenster auf einem einzigen Bildschirm zu verwalten, zwischen denen per Tastendruck hin- und hergesprungen werden kann.

Windows in Basic 7.0

Mit dem Befehl »WINDOW« kann man in Basic auf einfache Weise ein Fenster erzeugen.
Die Syntax ist

WINDOW links,oben,rechts,unten<,löschflag>

links = linke Grenze (Spalte 0-39 bzw. 0-79 im 80-Zeichen-Modus)
oben = obere Grenze (Zeile 0-24)

rechts = rechte Grenze (Wertebereich wie »links«)
unten = untere Grenze (Wertebereich wie »oben«)
löschflag = Flag, ob das Window auch gleich gelöscht werden soll (1 = löschen, 0 = nicht löschen). Wird »löschflag« weggelassen, nimmt das Basic 7.0 den Wert 0 an (0 = Flag für »nicht löschen«).

Man muß darauf achten, daß der Wert für »links« kleiner als der für »rechts« und der Wert »oben« kleiner als »unten« ist, damit die Anweisung einen Sinn ergibt.
Beispiel:

```
WINDOW 9,5,29,19,1
```

erzeugt ein Window mit folgenden Grenzen:

linke Grenze = 9
obere Grenze = 5
rechte Grenze = 29
untere Grenze = 19

Da das Flag = 1 ist, wird das Window auch gleichzeitig gelöscht.

Sehen wir uns noch ein kleines Beispielprogramm zum »Window«-Befehl an:
Beschreibung: Demo für Window-Programmierung in Basic 7.0
Filename: »'window'-demo«

```
100 REM *******************************
110 REM *                             *
120 REM *   DEMONSTRATIONSPROGRAMM    *
130 REM *                             *
140 REM *     ZUM BEFEHL "WINDOW"     *
150 REM *                             *
160 REM *******************************
170 :
180 SCNCLR
190 PRINT:PRINT "BITTE JETZT IM";CHR$(18);"EINGABE-FENSTER"
200 PRINT:PRINT "EINEN STRING (Z.B. NAMEN) EINGEBEN !"
210 CHAR,4, 6,"┌───────────────────────────────┐"
220 CHAR,4, 7,"│ E I N G A B E - F E N S T E R │"
230 CHAR,4, 8,"├───────────────────────────────┤"
240 CHAR,4, 9,"│                               │"
250 CHAR,4,10,"└───────────────────────────────┘"
260 :
270 CHAR,7,15,"*********************************"
280 CHAR,7,16,"* A U S G A B E - F E N S T E R *"
290 CHAR,7,17,"*********************************"
300 CHAR,7,18,"*                               *"
310 CHAR,7,19,"*********************************"
320 :
330 WINDOW 5,9,35,9:REM EINGABE-FENSTER OEFFNEN
340 :
350 OPEN 1,0:INPUT#1,A$:CLOSE 1
360 :
370 WINDOW 8,18,39,18:REM AUSGABE-FENSTER OEFFNEN
380 :
390 PRINT A$;
400 :
410 PRINT CHR$(19);CHR$(19):PRINT:PRINT:REM FENSTER AUFLOESEN
420 END
```

Dieses Programm fordert Sie zur Eingabe in einem von Zeile 330 definierten Eingabefenster auf. Ihre Eingabe wird dann noch einmal im Ausgabefenster, das von Zeile 370 erzeugt wird, angezeigt. Dann wird das Window durch zweifaches <HOME> (Zeile 410) aufgelöst und das Programm beendet (Zeile 420).
Stören Sie sich bitte nicht an den CHAR-Befehlen in den Programmzeilen 210-310; diese sind nur für die Bildschirmausgabe verantwortlich und somit in diesem Zusammenhang belanglos. Eine Beschreibung kommt in 3.4.5 im Rahmen des Kapitels 3.4, welches sich mit den Basic-7.0-spezifischen Befehlen, also auch CHAR, beschäftigt.
Zurück zum Demoprogramm.
Unser Programm »'window'-demo« soll nur eine Anregung darstellen; obwohl die Window-Technik des C128 mit der eines Commodore AMIGA oder Atari ST nicht mithalten kann, eröffnen sich viele Möglichkeiten, wenn man einfallsreich genug ist.
So ist es z.B. denkbar, in einem Programm die oberste und unterste Bildschirmzeile unabhängig vom Gesamtbildschirm für Hinweise einzusetzen. Dabei muß man folgendermaßen vorgehen:

- Mit »WINDOW 0,1,39,23« (im 80-Zeichen-Modus ist die »39« durch »79« zu ersetzen) die oberste und unterste Zeile ausklammern» (probieren Sie es im Direktmodus aus!).
- Ein Unterprogramm zum Schreiben einer Hinweiszeile muß zuerst mit »PRINT CHR$(19)CHR$(19);« (siehe Zeile 410 im Demoprogramm) den ganzen Bildschirm zugänglich machen und dann die Hinweiszeilen auf den Bildschirm bringen. Danach werden wieder durch den genannten WINDOW-Befehl die beiden Hinweiszeilen geschützt.

Nehmen Sie es mir bitte nicht übel, daß ich es bei dieser Anregung bewenden lassen muß; weil in diesem Buch unzählige Programme vorgestellt werden, sind durch die Kapazität der Programmdiskette und des Buches Grenzen gesetzt. Aufgrund seiner Detailliertheit dürfte das obige »Kochrezept« aber genügen.
Zum Abschluß dieses Kapitels 2.1.3 möchte ich Sie auf den Abschnitt 3.6.7 hinweisen, in dem eine andere Art der Window-Programmierung erläutert wird; diese setzt den WINDOW-Befehl nicht ein, sondern basiert vielmehr auf anderen Befehlen zur Bildschirmausgabe - lassen Sie sich überraschen!

2.2 Die zwei Bildschirme

Mehrfach wurde jetzt schon die Fähigkeit des C128, zwei(!) Bildschirme zur Ein-/Ausgabe zu benutzen, erwähnt.
Wie Sie wissen, kann der C128 einen Bildschirm darstellen, der über 40 Zeilen und 25 Zeilen verfügt wie der C64. Dies liegt daran, daß (fast) derselbe Chip dafür zuständig ist: der altbekannte VIC (s. C128-Handbuch, Anhang E, Seiten E-2 und E-3).
Zusätzlich kümmert sich ein anderer Video-Chip namens VDC (s. C128-Handbuch, Anhang E, Seiten E-4 bis E-7) um den 80-Zeichen-Bildschirm, welcher bei einer Kapazität von 80*25 = 2000 Zeichen mehr Text aufnimmt als der 40er mit 40*25 = 1000 Zeichen.

Da beide Video-Chips ein anderes Signal über eine andere Leitung senden, kommen sich der VIC und der VDC nicht in die Quere; jeder bezieht aus dem C128-Speicher aus unterschiedlichen Speicherbereichen die Information, was auf »seinem« Bildschirm abgebildet werden soll; diese setzt er in ein für den Monitor/Fernseher günstiges Format um und sendet dieses unaufhörlich. Dafür, was in den relevanten Speicherbereichen (Bildschirm- und Grafikspeicher) steht, ist die Zentraleinheit selbst verantwortlich. In der Regel wird das Betriebssystem (das zentrale Steuerprogramm, das automatisch nach dem Einschalten die Kontrolle über den C128 übernimmt) diese Aufgabe ausfüllen.
Aufgrund der unterschiedlichen Anlagen der beiden Bildschirme (der 80-Zeichen-Bildschirm hat doppelte Kapazität wie der 40er usw.) kann der C128 aber immer nur einen Bildschirm gleichzeitig für Ein-/Ausgabe verwenden. Der andere wird zwar vom zuständigen Video-Chip weiterhin angezeigt, aber der Cursor wird sich immer nur in einem einzigen Bildschirm befinden.
Wie man nun einen (oder zwei) Monitor(e) anschließt, möchte ich hier nicht behandeln, da ich davon ausgehe, daß Sie sich schon im Handbuch-Kapitel 1 darüber informiert haben (sonst könnten Sie Ihren C128 kaum betreiben).
Hier soll uns hauptsächlich interessieren, wie man als Anwender und Programmierer mit den zwei Bildschirmen effektiv arbeiten kann.
Zunächst sei noch einmal erwähnt, wie man den Darstellungsmodus auswählt, in dem der Computer Ein- und Ausgaben tätigen soll.
Die einfachste Möglichkeit ist, vor dem Einschalten des Geräts, vor einem Reset oder vor Betätigen von <RUN/STOP>+<RESTORE> die <40/80 DISPLAY>-Taste in die gewünschte Stellung zu bringen: <40/80 DISPLAY> verriegelt = 80 Zeichen, entriegelt = 40 Zeichen.
Will man später in den anderen Modus wechseln (vom 40-Zeichen-Modus in den 80er oder umgekehrt), löst man ESC-X aus, was schon in 2.1.2 angesprochen wurde. In einem Programm kann man ESC-X durch

```
PRINT CHR$(27);"X";
```

simulieren, um ein programmgesteuertes Umschalten durchzuführen.
Da nach ESC-X weiterhin beide Bildschirme dargestellt werden (der Video-Chip des Darstellungsmodus, der verlassen wurde, erzeugt schließlich nach wie vor das Bild am Monitor/Fernseher), stehen uns interessante Anwendungen offen, wie dieses Kapitel 2.2 zeigen will.

2.2.1 Der 80-Zeichen-Bildschirm

Da der 40-Zeichen-Modus schon vom C64 her bekannt ist, bedarf es eines kleinen Abschnittes mit Erklärungen zum 80-Zeichen-Modus.

Der Video-Chip VDC

Wie im C64, ist für die Erzeugung des 40-Zeichen-Bildschirms im C128 ein VIC (Video Interface Controller) eingebaut, der die gleichen Speicherbereiche wie beim C64 belegt:

Bildschirmspeicher: 1024 - 2023 (hexadezimal: $0400-$07E7)
Register-Basisadr.: 53248 (hexadezimal: $D000)
Farb-RAM für Text: 55296 - 56295 (hexadezimal: $D800-$DBE7)

Die einzige bedeutende Änderung ist, daß der VIC im C128 noch zwei zusätzliche Register hat, die allerdings in keinem Zusammenhang zu seiner eigentlichen Aufgabe, der Erzeugung des Bildschirms, stehen und in 5.2.3 besprochen werden, da sie im Zusammenhang mit dem in Kapitel 5 behandelten C64-Modus viel wichtiger sind.
Völlig neu hingegen ist der VDC. Dieser hat einen eigenen RAM-Bereich von 16K Kapazität (!), um möglichst wenig vom Hauptspeicher zu verbrauchen. Bedauerlicherweise sind alle diese RAM-Bereiche einschließlich der Register nicht direkt über PEEK und POKE ansprechbar (wie die VIC-Speicher), sondern nur über die Adressen 54784 und 54785. Über diese Adressen werden alle Schreib- und Lesezugriffe auf den VDC abgewickelt, was den Computer viel Rechenzeit kostet. Dies wird erst in 3.8 besprochen, wenn Sie sich aber gleich informieren wollen, sei Ihnen die Tabelle im Handbuch ab Seite E-4 (Anhang E, Seite 4) empfohlen; gleichzeitig möchte ich aber davon abraten, das dort beschriebene Verfahren zur VDC-Register-Manipulation anzuwenden. Die im Handbuch vorgestellte Methode ist erstens nicht fehlerfrei und zweitens wenig effektiv; in 3.8 entwickeln wir eine bessere Lösung.

Bildschirmorganisation im 80-Zeichen-Modus

Wenn Sie im 80-Zeichen-Modus versuchen, von Groß-/Grafik- auf Klein-/Groß-Schrift umzuschalten (SHIFT+CBM), werden Sie sehen, daß die Zeichen, die bereits am Bildschirm stehen, nicht umgestellt werden, wie Sie es vom C64 und dem 40-Zeichen-Modus gewöhnt sind. Geben Sie dann aber weitere Zeichen ein, werden diese im neuen Schriftmodus angezeigt. Der VDC ist nämlich in der Lage, zwei (!) Zeichensätze gleichzeitig darzustellen. Dies ist deswegen möglich, weil er sich zu jedem der 2000 Zeichen am Bildschirm folgende Informationen merkt:

1) Zeichencode
Dies ist ein Bildschirmcode, wie er im Bildschirmspeicher des VIC auch stehen würde, mit dem Unterschied, daß er sich im VDC-eigenen Speicherbereich befindet und auf diese Weise keinen Platz im Hauptspeicher belegt.

2) Attribut
Hierunter fallen eine Reihe von Informationen:
- Farbe (eine von 16 möglichen)
- Blinkmodus (entscheidet, ob ein Zeichen blinkt)
- Unterstreichmodus
- Reversmodus
- Zeichensatz (Groß-/Grafik-Schrift oder Klein-/Groß-Schrift)

Da der VIC nur die Farbinformation speichert, sind auf dem 40-Zeichen-Bildschirm die ESC- bzw. CONTROL-Kommandos für Blinken und Unterstreichen wirkungslos sowie immer nur ein Zeichensatz darstellbar. Der Reversmodus kommt beim VIC dadurch zustande, daß die

Bildschirmcode über 127 für die reversen Zeichen stehen; beim VDC ist dies zwar auch möglich, allerdings kann man auch das entsprechende Attribut setzen. Dies hätte den Vorteil, daß man die Codes über 127 mit völlig neuen Zeichen belegen kann, die dann sogar ihrerseits wieder revers dargestellt werden können (durch Setzen des Attributes); aus Gründen der Übereinstimmung von 40- und 80-Zeichen-Modus macht das Betriebssystem davon jedoch keinerlei Gebrauch.
Für eine erste Einführung wollen wir es zunächst hierbei belassen. Im weiteren Verlauf des Buches werden Sie aber immer wieder interessante Anwendungen des VDC-Chips kennenlernen und auch über dessen Programmierung einiges erfahren (3.8). Dabei wird auch erklärt, wie man eine Hardcopy (Bildschirmausdruck auf den Drucker) vom 80-Zeichen-Bildschirm programmiert oder auf das Bildschirm-RAM zugreift.

2.2.2 Ermittlung des aktuellen Bildschirms

Wenn ein Programm oft zwischen 40- und 80-Zeichen-Modus wechselt oder unmittelbar nach dem Programmstart feststellen will, in welchem Modus sich der C128 gerade befindet, steht dafür ein Basic-7.0-Befehl zur Verfügung:

```
RWINDOW (2)
```

enthält die Anzahl der Zeichen in einer Bildschirmzeile (also 40 oder 80) und wird wie andere Basic-Funktionen (LEN, STR$, VAL usw.) eingesetzt.
Folgende Zeile am Programmanfang bewirkt, daß das Programm nur im 80-Zeichen-Modus läuft:

```
10 IF RWINDOW (2) = 40 THEN PRINT "NUR IM 80-ZEICHEN-MODUS !":END
```

Solange Sie nur in Basic programmieren, ist diese Lösung durchaus praktikabel; in Maschinensprache läßt sich die RWINDOW-Funktion jedoch nur sehr umständlich umsetzen, da zuerst die Zahl 2 im Fließkommaformat in den FAC (Floating Point Accumulator = Fließkomma-Akkumulator) geschrieben, dann die RWINDOW-Routine aufgerufen und schließlich das Ergebnis ins Integerformat gewandelt werden müßte. Deshalb ist es sinnvoll, die Adresse 215 (hexadezimal: $D7), die den Wert 0 (40-Zeichen-Modus) oder 128 (80-Zeichen-Modus) enthält, abzufragen, wie es die RWINDOW-Routine des Basic-Interpreters tut. In Basic geht dies so:

```
10 IF PEEK (215)=0 THEN PRINT "NUR IM 80-ZEICHEN-MODUS"
```

Dies ist sehr leicht in Assembler übertragbar:

```
BIT $D7 ; Wert in 215 prüfen
```

Nach diesem Befehl ist das N-Flag (Negativ-Flagge) gesetzt, wenn der 80-Zeichen-Modus eingeschaltet ist. Ein BPL würde folglich im 40-Zeichen-Modus zu einer Abbruchroutine verzweigen.

Wenn Sie sich für die RWINDOW-Routine interessieren: Diese steht im ROM-Bereich von $8407 bis $8433. Den größten Durchblick verschafft man sich aber, indem man im »C128 ROM-Listing« aus der Commodore-Sachbuchreihe, vertrieben von Markt & Technik (Bestellnummer: MT 90212), auf Seite 205 nachschlägt.

2.2.3 Hochauflösende Grafik

Bekanntlich stehen für die Textdarstellung zwei Bildschirme zur Verfügung. Wie sieht es aber mit Grafiken aus?
Dazu ist zunächst einmal zu sagen, daß der 80-Zeichen-Bildschirm von Betriebssystem und Basic-Interpreter aus nur im Textmodus unterstützt wird, während für den 40-Zeichen-Bildschirm im Basic 7.0 eine Vielzahl komfortabler Grafikbefehle (s. 3.4.3) zur Verfügung steht. Die Grafikfähigkeiten des 80-Zeichen-Modus wurden leider unbeachtet gelassen; dieser bietet aber, da er ja doppelt so viele Zeichen darstellen muß, auch eine doppelt hohe Auflösung von stattlichen 640*200 Punkten (im Vergleich die Zahl des C64 und des C128 im 40-Zeichen-Modus: 320*200 Punkte).
Mit einem Hilfsprogramm, das in 3.7.1 vorgestellt wird, werden die Basic-7.0-Befehle dahingehend erweitert, daß sie auch den 80-Zeichen-Bildschirm für Hochauflösungsgrafiken nutzen können.
Wollen Sie schon einen Vorgeschmack bekommen? Dann gehen Sie bitte in den 80-Zeichen-Modus, geben Sie

```
RUN"GRAFIK-80.DEMO 2"
```

ein und lassen Sie die Programmdiskette im Laufwerk. Folgen Sie dann nur noch der Anweisungen des Programms, <RETURN> zu drücken.
Eine Eigenschaft des VIC, die für Spieleprogrammierung sehr wichtig ist, ist aber trotz allem nicht im 80-Zeichen-Modus realisierbar, da der VDC (Video-Chip für den 80-Zeichen-Modus) dazu von seiner Hardware aus nicht in der Lage ist: die vom C64 her so beliebten Sprites.
Da der 80-Zeichen-Modus aber eher für »ernsthafte« Anwendungen wie Textverarbeitung, Dateiverwaltung, Geschäftsbilanzen usw. gedacht ist, während der 40-Zeichen-Modus für Spieleprogrammierung sehr nützlich ist, fällt dieser kleine Mangel nur minimal ins Gewicht.

2.2.4 Anwendungen der zwei Bildschirme

In diesem Unterkapitel werden drei Möglichkeiten vorgestellt, die beiden Bildschirme des C128 sinnvoll einzusetzen. Dabei ist Voraussetzung, daß Sie sowohl einen 40- als auch einen 80-Zeichen-Monitor haben (oder einen Monitor mit Umschaltmöglichkeit, wie etwa den Commodore 1901). Ist dies nicht der Fall, so lesen Sie bitte trotzdem diesen Abschnitt durch, denn einige Informationen sind auch bei Verwendung eines einzigen Bildschirms verwendbar.

Verwendung eines Bildschirms als Merkzettel

Die Window-Technik erlaubt es, bestimmte Bereiche des Bildschirms - durch Definition des Restbildschirms als Window - zu schützen und auf diese Weise als »Merkzettel« zu benutzen. Dies wollen wir einmal ausprobieren. Löschen Sie bitte den aktuellen Bildschirm (ob 40- oder 80-Zeichen-Modus, spielt hier keine Rolle) und schreiben Sie einen Text in die oberste(n) Bildschirmzeile(n). Drücken Sie dann <SHIFT>+<RETURN>, um an den Anfang der nächsten Zeile zu gelangen, und lösen Sie dann ESC-T aus (linke obere Ecke des Bildschirmfensters setzen).
Dann können Sie, solange Sie nicht <HOME><HOME> eingeben, Bildschirmoperationen jeder Art durchführen, ohne daß Ihr Text im obersten Teil des Bildschirms verlorengeht.
Hierfür gibt es eine Vielzahl von Anwendungen mit praktischem Nutzen; z.B. könnte man eine wichtige Notiz außerhalb des Windows anbringen, die das Programmieren erleichtert.
Der Nachteil ist jedoch, daß ein Teil des Bildschirms für die Notiz aufgewendet werden muß. Es ist aber möglich, mit ESC-X in den anderen Bildschirm zu springen, dort die Notiz abzulegen und wieder mit ESC-X in den ursprünglichen Bildschirm zurückzukommen. Dann verwenden Sie beispielsweise den 40-Zeichen-Bildschirm für Notizen und den 80-Zeichen-Bildschirm zum Programmieren (oder umgekehrt).
Auf diese Weise müssen Sie auf keinen Bildschirmplatz zum Programmieren verzichten und haben einen kompletten Bildschirm für mehr oder weniger umfangreiche Notizen (Programmlistings etc.) zur Verfügung.
Probleme ergeben sich jedoch bei Verwendung des FAST-Modus, der im Anschluß als Fall b) besprochen wird.

FAST-Modus

Der C128 kann, wie der Sinclair ZX 81, dadurch beschleunigt werden, daß man den Basic-Befehl FAST eingibt. Dann arbeitet er doppelt so schnell (für Profis: die Taktfrequenz wird von 1 von 1 Mega-Hertz auf 2 MHz erhöht), schaltet allerdings den 40-Zeichen-Bildschirm ab. Solange man mit 80 Zeichen pro Zeile arbeitet, dürfte dies nicht viel ausmachen, doch bei Verwendung des 40-Zeichen-Bildschirms ist der FAST-Befehl nur in zeitkritischen Programmteilen (Sortierroutinen etc.) geeignet.
Mit SLOW wird wieder der Ausgangszustand hergestellt.
Will man jedoch wegen der Merkzettel-Technik aus Anwendung a) beide Bildschirme gleichzeitig sehen, muß man auf den FAST-Modus verzichten.
Arbeitet man hingegen nur im 80-Zeichen-Modus, sollte man unmittelbar nach Einschalten, Reset oder <RUN/STOP>+<RESTORE> den Befehl FAST eingeben, da der 80-Zeichen-Modus sonst zu langsam arbeitet. Es ist auch in diesem Fall sinnvoll, den FAST-Befehl auf eine Funktionstaste zu legen, z.B. KEY 4, »FAST« + CHR$(13).

Grafikprogrammierung

Diese Anwendung ist wichtig, wenn Sie die Grafikbefehle des Basic 7.0 ausschöpfen wollen und Grafikprogramme in Basic schreiben.

Falls Ihnen momentan noch die entsprechenden Kenntnisse fehlen, überfliegen Sie diese Anwendung c) und lesen Sie sie dann, wenn Sie mit der Grafikprogrammierung über Basic 7.0 beginnen wollen.
In 2.2.3 haben wir bereits festgestellt, daß der über Basic erreichbare Grafikbildschirm auf dem 40-Zeichen-Bildschirm dargestellt wird.
Nach Ausführung von »GRAPHIC 1« wird der Textbildschirm (40 Zeichen) ausgeblendet und statt dessen der Grafikbildschirm angezeigt. Zunächst ist es dann ohne Bedeutung, ob sich der C128 vor dem GRAPHIC-Befehl im 40- oder im 80-Zeichen-Modus befand.
Dies kommt erst zum Tragen, wenn das Programm beendet wird (Fehlermeldung etc.), während der Grafikbildschirm eingeschaltet ist. Dann nämlich wird in den Textmodus vor »GRAPHIC 1« zurückgesprungen. Beim 80-Zeichen-Bildschirm bleibt der Grafikbildschirm am 40er erhalten; beim 40-Zeichen-Bildschirm schaltet der Computer jedoch vorher den Grafikmodus aus, damit der Text (Fehlermeldung etc.) sichtbar ist.
Dies läßt sich am leichtesten erklären, wenn Sie folgende Zeile im Direktmodus eingeben (1x im 40-, 1x im 80-Zeichen-Modus):

```
GRAPHIC 1,1:DRAW,0,0 TO 100,100:SLEEP 1:X
```

Zuerst wird die hochauflösende Grafik eingeschaltet, dann eine Linie gezeichnet, 1 Sekunde gewartet und zuletzt durch »X«, einen unsinnigen Befehl, ein »?SYNTAX ERROR« erzeugt.
Haben Sie die Zeile im 40-Zeichen-Modus eingegeben, wird zuerst die Grafik angezeigt, dann aber gelöscht, damit Sie die Fehlermeldung sehen; im 80-Zeichen-Modus aber bleibt die Grafik nach wie vor eingeschaltet, da sich der Text und die hochauflösende Grafik, die sich auf zwei unterschiedlichen Bildschirmen befinden, nicht gegenseitig stören.
Bei Grafikprogrammen ist es also sinnvoll, das Programm aus der 80-Zeichen-Darstellung zu starten, da dann Grafik und Fehlermeldung zusammen eine schnelle Lokalisierung des Fehlers ermöglichen.
Eine andere Variante, die vor allem dem Endanwender viel Freude bereitet, ist die gleichzeitige Anzeige von Daten am 80- und Grafik am 40-Zeichen-Bildschirm in einem Kalkulationsprogramm o.ä. Ein Beispiel hierfür wird sich in 3.4.3 ergeben; derzeit fehlen uns noch entsprechende Kenntnisse der Basic-7.0-Befehle.

2.3 Der deutsche Zeichensatz

Nun ist es an der Zeit, endlich den deutschen Zeichensatz (DIN-Zeichensatz) des C128 genauer zu besprechen, nachdem auf dieses Unterkapitel so oft hingewiesen wurde.
Dabei muß als erstes festgehalten werden, mit welcher Taste die Auswahl zwischen ASCII- und DIN-Zeichensatz gewählt wird. Die entsprechende Taste heißt in der ursprünglichen Version des C128 <CAPS LOCK> und wurde schon kurz erwähnt. Beim 128D (für diejenigen, denen diese Bezeichnung nichts sagt: das ist der C128 mit eingebautem Diskettenlaufwerk C1571, der dem AMIGA oder den PCs so ähnlich sieht) heißt diese Taste allerdings <ASCII/DIN>,

obwohl sich außer der Tastenbeschriftung nichts geändert hat. Der Einfachheit halber werde ich im folgenden nur immer von <CAPS LOCK> reden; wenn Ihr Gerät diese Taste nicht hat, so ist natürlich <ASCII/DIN> gemeint.
Als Rechtfertigung: Die 128D-ler sollen bestimmt nicht benachteiligt werden (ich arbeite selbst mit einem 128D), aber die »Tradition« spricht für die Bezeichnung <CAPS LOCK>.
Bevor wir anfangen, noch eine kleine Merkhilfe zu den Bezeichnungen ASCII-Zeichensatz/ DIN-Zeichensatz:
»ASCII« beginnt mit »A« wie »Amerikanischer Zeichensatz«,
»DIN« beginnt mit »D« wie »Deutscher Zeichensatz«.
Der Vollständigkeit halber: ASCII steht für »American Standard Code for Information Interchange« (amerikanischer Standardcode zum Informationsaustausch), DIN heißt »Deutsche Industrie-Norm« und wird deshalb für den deutschen Zeichensatz verwendet, weil bei gedrückter <CAPS LOCK>-Taste die genormte Schreibmaschinentastatur nach DIN simuliert wird.

CAPS LOCK - So funktioniert's

Wenn man die Funktionsweise von <CAPS LOCK> betrachtet, stellt man fest, daß diese eigentlich recht primitiv ist: sobald <CAPS LOCK> verriegelt ist, wird in dem Bereich, in dem der normale Zeichensatz als Muster untergebracht ist, statt des ASCII-Musters das DIN-Muster eingeblendet, d.h. der VIC stellt augenblicklich andere Zeichen am 40-Zeichen-Bildschirm dar. Dies erkennt das Betriebssystem dann und aktiviert eine entsprechende Tastaturtabelle. Dies hat zur Folge, daß sich die Tastenbelegungen der DIN-Norm anpassen (z.B. werden <Y> und <Z> vertauscht). Damit aber auch der 80-Zeichen-Bildschirm umgestellt wird, muß der gesamte deutsche Zeichensatz in den VDC-eigenen Speicher kopiert werden. Da das VDC-RAM nicht über einfache Schreibzugriffe, sondern nur über komplizierte Operationen angesprochen werden kann, dauert dieser Kopiervorgang verhältnismäßig lang. Das macht sich in der kleinen Pause, die bei Verriegeln von <CAPS LOCK> entsteht, bemerkbar. Diese Verzögerung kann man gut sehen, wenn man ein längeres Programm listet und mittendrin (beim Listen) <CAPS LOCK> verriegelt.
Einige Informationen darüber für die Profis unter Ihnen. Zuerst die beteiligten Adressen: der VIC-Zeichensatz liegt im Bereich 53248-56343 (hexadezimal $D000-$DFFF, 4K-ROM namens »CHARGEN«=»Character Generator«). Das Umschalten in diesem Bereich erfolgt über Bit 6 in Adresse 1 (Prozessorport), das gesetzt ist, wenn <CAPS LOCK> gedrückt ist, gelöscht, wenn <CAPS LOCK> entriegelt ist. So kann man das Drücken von <CAPS LOCK> auch simulieren, wie wir bald sehen werden.
Da aber nur der Zeichensatz an der Originaladresse ($D000 = dezimal 53248) vertauscht wird, ist <CAPS LOCK> für den Fall, daß ein eigener Zeichensatz an anderer Adresse (zum Beispiel $3000 = dezimal 12288) eingeschaltet wurde, wirkungslos, da sich der VIC seine Informationen nicht mehr aus dem $D-Bereich holt und somit ein Verändern desselben keine Bedeutung mehr hat.

Die Tastaturbelegung nach DIN und die Akzenttaste

Da der deutsche Zeichensatz seine eigene Tastaturbelegung hat, sind auf der Tastatur die Ver-

änderungen gegenüber dem ASCII-Zeichensatz weitgehend in grauer Farbe neben der schwarzen Normalbeschriftung eingetragen, weshalb die Umstellung nicht schwierig ist. Eine Besonderheit ist jedoch die Akzenttaste, die sonst mit dem Minus-Zeichen »-« belegt ist. Diese ist bezüglich der Art der Eingabe mit <ESC> vergleichbar; nach Drücken der Akzenttaste bewegt sich der Cursor noch nicht, es erscheint aber der entsprechende Akzent (mögliche Akzente erreichbar über: <->, <SHIFT>+<-> und <CBM>+<->). Der nächste Tastendruck ergibt dann das endgültige Zeichen, wofür es zwei Möglichkeiten gibt:

- Das Zeichen ergibt mit dem Akzent eine sinnvolle Kombination (z.B. akzentuiertes »e« im Kleinschriftmodus funktioniert bei jedem der drei Akzente; probieren Sie es deshalb bitte der Übung halber aus).
- Andernfalls wird die gedrückte Taste wie gewohnt bearbeitet, als ob vorher kein Akzent ausgelöst worden wäre.

Es sei darauf hingewiesen, daß die Akzenttaste natürlich nur im DIN-Zeichensatz funktioniert, wobei die meisten akzentuierten Zeichen nur im Kleinschriftmodus erzeugt werden können.

Da aber nicht alle Besonderheiten der DIN-Tastaturbelegung auf der Tastatur eingetragen sind, seien noch ein paar interessante Kombinationen erwähnt, die man nicht so leicht ausknobelt:

- den Klammeraffen (@) erreicht man über <CBM> und die herkömmliche <@>,
- das griechische »My«, das bei Maßeinheiten die Abkürzung für »Mikro« ist, über <CBM> und den Linkspfeil »←«,
- das griechische »Sigma«, das für Statistik wichtig ist, über <CBM> und die Taste, die auf der ASCII-Tastatur das Größer-Zeichen »>« darstellt,
- das mathematische Radizierungssymbol (Wurzelzeichen) über <CBM> und den sonstigen Schrägstrich »/«.

Deutscher Zeichensatz mit PEEK und POKE

Wie schon angekündigt, kann man das Drücken der <CAPS LOCK>-Taste durch Löschen von Bit 6 im Prozessorport, der Adresse 1, simulieren. Vorher muß jedoch das Datenrichtungsregister des Prozessorports, die Adresse 0, so geschaltet werden, daß ein Zugriff auf die Adresse 1 möglich ist. Dies beides bewerkstelligen folgende zwei POKEs:

```
POKE 0,PEEK (0) OR 64:POKE 1,PEEK (1) AND 191
```

Dann wird auf den DIN-Zeichensatz geschaltet; dieser bleibt auch bei noch so heftigem Ver- und Entriegeln von <CAPS LOCK> erhalten. Abhilfe schafft nur der POKE-Befehl

```
POKE 1,PEEK (1) OR 64
```

Dieser schaltet auf ASCII (vorher ist kein POKE ins Datenrichtungsregister nötig); allerdings besteht dann wieder die Umschaltmöglichkeit über <CAPS LOCK>. Der folgende Abschnitt wird aber eine Lösung vorstellen, wie man das Umschalten zumindest im 80-Zeichen-Modus verhindern kann.

2.3.1 CAPS LOCK inaktivieren

Oft will man ein Programm gegen bestimmte Eingriffe durch den Anwender schützen, z.B. indem man einen Wechsel des Zeichensatzes verhindert.
Solange man den deutschen Zeichensatz über POKE einschaltet, hat man einen hervorragenden Schutz, da nur ein POKE-Befehl wieder auf den ASCII-Zeichensatz schaltet, während <CAPS LOCK> wirkungslos ist.
Schwierig wird es aber, wenn man den ASCII-Zeichensatz absichern will.
Der entsprechende POKE-Befehl schaltet zwar um, kann aber nicht dafür sorgen, daß <CAPS LOCK> wieder in den deutschen Zeichensatz springt. Der Grund ist, daß die <CAPS LOCK>-Taste »low-aktiv« ist, was konkret heißt, daß man den »Entriegelt«-Zustand nicht absichern kann.
Im 80-Zeichen-Modus wirkt aber ein POKE, der das Betriebssystem dahingehend beeinflußt, daß es die <CAPS LOCK>-Taste im 80-Zeichen-Modus nicht berücksichtigt:

```
POKE 2757,129
```

Dies kann man über folgenden POKE-Befehl rückgängig machen:

```
POKE 2757,0
```

Im 40-Zeichen-Modus ist dieser POKE jedoch ungeeignet. Es bleibt für den Fall, daß in der 40-Zeichen-Darstellung der ASCII-Zeichensatz nicht mehr umgeschaltet werden soll, nur die sehr umständliche Methode, den Zeichensatz an eine andere Adresse im Speicher zu kopieren und den VIC zu veranlassen, seine Informationen über das Aussehen der Zeichen am Bildschirm aus der Zeichensatzkopie zu entnehmen. Dieses Verfahren funktioniert, da - wie bereits erwähnt - ein Zeichensatz, der nicht an der Originaladresse (53248 = $D000) liegt, durch <CAPS LOCK> nicht beeinflußbar ist.
Allerdings ist die Umsetzung in eine lauffähige Routine recht umständlich und zudem sehr stark von bestimmten Eigenheiten jedes einzelnen Programms abhängig, weshalb die Beschreibung des Verfahrens genügen muß. Wer in der Lage ist, dies zu programmieren, wird dabei keine Schwierigkeiten haben; andere sollten von der heiklen Thematik »Zeichensatz-Verschieben« lieber ihre Finger lassen. Falls Sie sich für solche Grafikfragen interessieren, sei Ihnen das Buch »Grafikprogrammierung C128« von Heimo Ponnath, erschienen im Markt & Technik Verlag, Nummer MT 90202, empfohlen. Dort wird auch das Verschieben und Modifizieren des Zeichensatzes ausführlich besprochen.

2.3.2 Vergleich der Zeichenmatrix

Da der C128 über zwei verschiedene Zeichensätze verfügt, ist ein Vergleich der Zeichenmatrix interessant. Die Zeichenmatrix ist, vereinfacht gesagt, das Zeichenmuster. So wie ein Sprite durch ein Muster definiert wird, das in Form von Bits und Bytes im Speicher steht, ist auch jedes einzelne Zeichen definiert. Die Gesamtheit dieser Definitionen aller Zeichen heißt dann »Zeichensatz«.

Um nebeneinander zu einem Code sowohl die Zeichenmatrix im ASCII-als auch die im DIN-Zeichensatz anzuzeigen, benötigt man ein kleines Hilfsprogramm.
Beschreibung: Vergleich der Matrix von ASCII- und DIN-Zeichensatz
Filename : »matrix-vergleich«

```
100 REM ********************************
110 REM *                              *
120 REM * VERGLEICH DER ZEICHENMATRIX *
130 REM *                              *
140 REM * ZWISCHEN AMERIKANISCHEM UND *
150 REM *                              *
160 REM *    DEUTSCHEM ZEICHENSATZ     *
170 REM *                              *
180 REM ********************************
190 :
200 BANK 14:REM ZEICHENGENERATOR MITTELS BANK-SWITCHING ZUGAENGLICH MACHEN
210 :
220 IF RWINDOW(2)=80 THEN FAST:REM DOPPELTE GESCHWINDIGKEIT IM 80-ZEICHEN-MODUS
230 :
240 DIM AM$(7),DT$(7):REM ZEICHENMATRIX VON AMERIKANISCHEM UND DEUTSCHEM ZEICHEN
SATZ KOMMT IN DIESE BEIDEN ARRAYS
250 :
260 DEF FN BC(A)=A+33*(A=255)+64*(A>63)+32*(A<96)-32*(A<160)+64*(A>191):REM ****
 FUNKTION ZUR UMRECHNUNG EINES ASCII-CODES IN BILDSCHIRMCODE
270 DEF FN AC(B)=B+64+64*(B<64 AND B>31)+32*(B<96 AND B>63):REM *** FUNKTION ZUR
 UMRECHNUNG EINES BILDSCHIRMCODES IN ASCII-CODE
280 :
290 :
300 P$=CHR$(18)+CHR$(32)+CHR$(146):REM REVERS ON, SPACE, REVERS OFF (REVERSDARST
ELLUNG EIN, LEERZEICHEN, REVERSDARSTELLUNG AUS)
310 :
320 SCNCLR:PRINT "DRUECKEN SIE NUN ..."
330 PRINT "<Z>   BEI EINGABE DES ZEICHENS";CHR$(13);
340 PRINT "<A>   BEI EINGABE DES ASCII-CODES";CHR$(13);
350 PRINT "<B>   BEI EINGABE DES BILDSCHIRMCODES";CHR$(13)
360 :
370 DO:GETKEY A$:LOOP UNTIL A$="Z" OR A$="A" OR A$="B"
380 PRINT ":";A$
390 :
400 IF A$="A" THEN BEGIN
410 :INPUT "ASCII-CODE";A
420 :IF A<32 OR A>255 OR A<>INT(A) OR (A>127 AND A<160) THEN 410:REM ***********
 UNERLAUBTE EINGABEN (ALSO EINGABEN OHNE SINNVOLLES ERGEBNIS) ZURUECKWEISEN
430 :INPUT "REVERS (J/N)";J$
440 :B=FN BC(A) OR (-128)*(J$="J"):REM BILDSCHIRMCODE ERRECHNEN (BEI REVERSDARST
ELLUNG WIRD 128 ADDIERT, ANSONSTEN IST DER BILDSCHIRMCODE = FN BC(A)
450 :PRINT "BILDSCHIRMCODE:";B
460 BEND
470 :
480 IF A$="B" THEN BEGIN
490 :INPUT "BILDSCHIRMCODE";B
500 :IF B<0 OR B>255 OR B<>INT(B) THEN 490
510 :PRINT "ASCII-CODE:";FN AC(B AND 127);:REM ASCII-CODE MITTELS FN AC(B) ERREC
HNEN
520 :IF B>127 THEN PRINT "(REVERS)":ELSE PRINT:REM GGF. "(REVERS)" AUSDRUCKEN
530 BEND
540 :
550 IF A$="Z" THEN BEGIN
560 :INPUT "ZEICHEN";Z$
570 :IF LEN(Z$)<>1 THEN 560
580 :IF ASC(Z$)<32 OR (ASC(Z$)>127 AND ASC(Z$)<160) THEN 560
```

```
590 :INPUT "REVERS (J/N)";J$
600 :B=FN BC(ASC(Z$)) - 128*(J$="J"):REM BEI REVERSDARSTELLUNG 128 ADDIEREN
610 :PRINT "BILDSCHIRMCODE:";B
620 BEND
630 :
640 :
650 INPUT "KLEINSCHRIFT-MODUS (J/N)";JN$:B=B-256*(JN$="J")
660 :
670 :
680 PRINT CHR$(13);CHR$(13)
690 PRINT "AMERIKANISCH       DEUTSCH"
700 PRINT "------------------------------------";CHR$(13)
710 :
720 POKE 0,PEEK(0) AND 63:REM AMERIKANISCHEN ZEICHENSATZ EINSCHALTEN
730 :
740 FOR F=0 TO 7
750 :BYTE=PEEK(53248+8*B+F):AM$(F)=""
760 :FOR BIT=7 TO 0 STEP -1:REM ALLE 8 BITS (BITS WERDEN MIT 0 BIS 7 NUMERIERT)
DURCHCHECKEN (DABEI WIRD RUECKWAERTS GEZAEHLT; WESHALB STEHT IM BUCH!)
770 ::IF BYTE AND 2↑BIT THEN AM$(F)=AM$(F)+P$:ELSE AM$(F)=AM$(F)+" "
780 :NEXT BIT
790 NEXT F
800 :
810 POKE 0,PEEK(0) OR 64:POKE 1,PEEK(1) AND 63:REM DEUTSCHER ZEICHENSATZ EIN
820 :
830 FOR F=0 TO 7
840 :BYTE=PEEK(53248+8*B+F):DT$(F)=""
850 :FOR BIT=7 TO 0 STEP -1
860 ::IF BYTE AND 2↑BIT THEN DT$(F)=DT$(F)+P$:ELSE DT$(F)=DT$(F)+" "
870 :NEXT BIT
880 NEXT F
890 :
900 POKE 0,PEEK(0) AND 63
910 :
920 FOR F=0 TO 7:REM AUSGABE DER MATRIZEN ("MATRIZEN" = MEHRZAHL VON "MATRIX")
930 :PRINT CHR$(32);AM$(F);SPC(8);DT$(F)
940 NEXT F
950 :
960 PRINT:PRINT:GOTO 330
```

Dieses Programm macht ausgiebigen Gebrauch von Basic-7.0-Befehlen und ist deshalb, solange Sie sich in das C128-Basic noch nicht eingearbeitet haben, schwer durchschaubar. Dies sollte aber nicht allzu sehr stören, da »MATRIX-VERGLEICH« zunächst nur für Anwendungszwecke vorgesehen ist.

Bedienung von »MATRIX-VERGLEICH«

Die Bedienung des Programms ist aufgrund der Menüsteuerung und der wenigen benötigten Parameter schnell erklärt.

»MATRIX-VERGLEICH« arbeitet auf beiden Bildschirmen, also sowohl im 40- als auch im 80-Zeichen-Modus; die 40-Zeichen-Darstellung ist allerdings vorzuziehen, da die Matrix größer und deutlicher als am 80-Zeichen-Bildschirm ist.

Die einzigen Informationen, die das Programm benötigt, sind die genauen Parameter des Vergleichs-Zeichens.

Das Zeichen selbst können Sie direkt per Tastatur eingeben, wenn Sie nach Anwahl von <Z> (im Hauptmenü) die entsprechende Taste und <RETURN> betätigen. Wenn Sie das Revers-

Muster wünschen, geben Sie noch <J> ein (sonst <N>). Bei jeder Art von Parametereingabe aus dem Hauptmenü heraus fragt der C128, ob Sie das Zeichen im Groß-/Grafik- oder im Klein-/Groß-Zeichensatz haben wollen. Für Kleinschrift müssen Sie <J> eingeben, für Groß/ Grafik <N>.
Die Eingabe des ASCII-Codes (<A> im Hauptmenü) läuft entsprechend ab, wobei Sie anstelle des Zeichens den ASCII-Code desselben mitteilen müssen.
Bei Eingabe des Bildschirmcodes (<B> im Hauptmenü) wird die Reversinformation nicht extra abgefragt, da diese im Bildschirmcode selbst enthalten ist:

Bildschirmcode < 128 (Codes 0-127): nicht revers
Bildschirmcode >=128 (Codes 128-255): revers

Nach der Parametereingabe wird das Zeichen links am Bildschirm im amerikanischen, rechts im deutschen Zeichensatz angezeigt; danach befinden Sie sich automatisch wieder im Hauptmenü.

Programmbeschreibung

Wenn Sie sich die Basic-7.0-Befehle angeeignet haben, können Sie diese Programmbeschreibung verstehen. Ansonsten machen Sie bitte mit dem nächsten Kapitel weiter, wo dieses Thema erläutert wird. Danach können Sie immer noch kurz zurückblättern.
Die ersten Zeilen (bis Zeile 290) dienen der Initialisierung. Der Zeichengenerator wird zugänglich gemacht, im 80-Zeichen-Modus wird auf doppelte Geschwindigkeit geschaltet (siehe 2.2.2 und 2.2.4), die nötigen Arrays (Variablenfelder) werden dimensioniert und zwei Funktionen zur Umrechnung von ASCII- in Bildschirm-Code und umgekehrt werden definiert (die Funktionsweise dieser raffinierten Funktionen wird in 3.3 besprochen).
In Zeile 300 wird ein String P$ (P$ steht für »Punkt«) vorbereitet, der ein reverses Quadrat erzeugt, das in der Zeichenmatrix als gesetzter Punkt dient. Hauptmenü und Parametereingabe stehen in den Zeilen 310-670, wo die Position der Zeichenmatrix im Zeichengenerator-ROM Schritt für Schritt errechnet wird und in B steht. Dabei werden einige unsinnige Eingaben zurückgewiesen (Zeilen 420, 500, 570/580).
Als nächstes wird dann die Überschrift für die Zeichenmatrizen ausgegeben (680-710).
Mit Zeile 720 beginnt die Berechnung der Zeichenmatrix und die Ablage in den Arrays AM$() und DT$().
Ab Zeile 920 erfolgt nur noch die Ausgabe der Zeichenmatrizen nebeneinander (920-940) und der Rücksprung ins Hauptmenü (960).

3
Von Basic 2.0 zu Basic 7.0 - und noch weiter

Das mächtige Basic 7.0 des C128 ist eine erhebliche Verbesserung gegenüber dem Basic 2.0 des C64, allerdings ist es auch keine völlig neue Sprache. Dieses große Kapitel führt Sie anhand von Beispielen in die Arbeit mit Basic 7.0 ein, steht Ihnen mit Ratschlägen, Tips und Tricks zur Seite und wird Ihnen das neue Basic vertraut machen.
Dabei wird es jedoch nicht bleiben. Sie bekommen neben vielen Utilities (Hilfsprogrammen) auch Informationen darüber, wie man C64-Basicprogramme in Basic 7.0 umschreibt, um sie im C128-Modus laufen zu lassen, und werden vor allem durch Anwendungen unterstützt. Darunter fallen sowohl Grafikprogramme als auch Routinen, die Ihnen das Verständnis erleichtern oder einfach Programmierarbeit abnehmen.
Ferner finden Sie Kurse über die Programmierung von Eingabemasken, komfortablen Menüs der Extraklasse und Windows, damit Sie Ihre eigenen Programme professioneller gestalten können, ohne nennenswerte Mehrarbeit zu investieren.
Erweiterungen zum Basic 7.0, die Sie direkt von Diskette laden können, um effektiver zu arbeiten, werden auch ausführlich erklärt. Ein Unterkapitel mit Tips & Tricks zum Basic 7.0, in dem mehr als nur eine Reihe von PEEKs, POKEs und SYS-Befehlen steht, die das Letzte aus Ihrem C128 herauskitzeln, bildet dann den Abschluß. Sie sehen, wir haben eine Menge vor: Fangen wir an.
Deshalb möchte ich Sie bitten, Ihr C128-Handbuch bereitzulegen. Dieses enthält nämlich ausgezeichnete Erklärungen zu allen Basic-7.0-Befehlen, weshalb sich eine Wiederholung oder Neufassung erübrigt. Außerdem erwarten Sie von diesem Buch völlig zu Recht neue Informationen und Hilfestellungen und nicht alte Erkenntnisse in neuer Formulierung.

3.1 Vergleich der Zeichencodes von C64 und C128

Bei den C128-Zeichensätzen (deutsch/amerikanisch) wird zwischen DIN- und ASCII-Zeichensatz unterschieden. Der amerikanische ASCII-Zeichensatz darf aber nicht mit dem ASCII-Code eines Zeichens verwechselt werden.
Mit »ASCII-Zeichensatz« ist lediglich der amerikanische Zeichensatz des C128 (im Gegensatz zum deutschen DIN-Zeichensatz) gemeint, also der Zeichensatz, den auch der C64 hat (keine Umlaute usw.).
»ASCII-Code« nennt man den Code eines Zeichens, der bei PRINT CHR$(x) für »x« eingesetzt werden muß, damit das entsprechende Zeichen am Bildschirm erscheint. Den ASCII-Code eines Zeichens erhält man am einfachsten über PRINT ASC(»Zeichen«). Beispiel: PRINT ASC(»A«) ergibt 65, den ASCII-Code des Zeichens »A«.
Eine Tabelle aller Zeichen des C128 und ihrer ASCII-Codes finden Sie übrigens im Anhang A ab Seite A-7. Die Codetabelle des C64 gilt auch im C64-Modus und beginnt, nicht weit von der C128-Tabelle entfernt, auf der Seite A-1.

3.1.1 Bildschirmcodes

Auch im C64-Handbuch werden die Bildschirmcodes erklärt. Die dortigen Ausführungen wurden in das C128-Handbuch übernommen und beginnen dort auf Seite 5-80 (Seite 80 von Kapitel 5). Ab Seite A-4 (Seite 4 im Anhang A) finden Sie noch eine kleine Zusammenfassung und schließlich die Tabelle der Bildschirmcodes, die ebenfalls dem C64-Handbuch entstammt.
Ein Vergleich der Bildschirmcodes von C64 und C128 bringt das eindeutige Ergebnis, daß im amerikanischen Zeichensatz exakt dieselben Bildschirmcodes wie beim C64 verwendet werden. Der Grund dafür ist, daß in der 40-Zeichen-Darstellung der VIC, also der gleiche Chip wie im C64 zuständig ist und der 80-Zeichen-Chip VDC zum VIC bildschirmcode-kompatibel ist.
Die Bildschirmcodes im DIN-Zeichensatz sind allerdings, wie der DIN-Zeichensatz überhaupt, eine Neuerung, da der C64 nur über den ASCII-Zeichensatz verfügt. Deswegen werden wir uns mit diesem Problem ausführlicher beschäftigen müssen.
Beachten Sie bitte, daß sich alles von jetzt an bis (ausschließlich) 3.1.2 Gesagte nur auf den DIN-Zeichensatz bezieht (sofern nicht anders angegeben). Der ASCII-Zeichensatz unterscheidet sich bekanntlich nicht vom C64-Zeichensatz, weshalb sich eine längere Besprechung erübrigt.

Bildschirmcodes im DIN-Modus

Die Codes der Zeichen, die nur ihr Aussehen verändert haben, aber nicht neu hinzugekommen sind, stimmen mit denen im ASCII-Zeichensatz überein. Dies gilt für alle Buchstaben (die Y-Z-Vertauschung auf der Tastatur wirkt sich auf den Bildschirmcode überhaupt nicht aus), Zahlen und Symbole (Zeichen wie »>«, »↑«, »;« usw.) mit wenigen Ausnahmen (Klammeraffe »@«, Pfund-Symbol »£« und Linkspfeil »←«).

Zur Demonstration geben Sie bitte folgende Zeile im Direktmodus bei entriegelter <CAPS LOCK>-Taste ein:

```
FOR I = 33 TO 95:PRINT CHR$(I);:NEXT I
```

Diese Zeile schreibt die genannten Zeichen auf den Bildschirm. Dann gehen Sie bitte in den DIN-Zeichensatz. Wenn Sie genau hinsehen, wird Ihnen auffallen, daß die meisten Zeichen lediglich Ihre Form ändern, aber die Bedeutung geblieben ist; die Buchstaben sind jetzt beispielsweise etwas anders geformt, aber sie sind nicht zu völlig anderen Zeichen geworden. Gleiches gilt für die Zahlen und die meisten Symbole.
Einige Zeichen fallen jedoch auf: der Klammeraffe wird zum Paragraph, das Pfund-Symbol zum Schrägstrich und der Linkspfeil zum Querstrich. Dies sind die erwähnten Ausnahmen.

Bildschirmcodes für Grafikzeichen im DIN-Modus

Da pro Zeichensatz nur 128 verschiedene Zeichen (und 128 Reverszeichen) möglich sind, konnte Commodore nicht ohne weiteres den ASCII-Zeichensatz um weitere Zeichen erweitern, sondern mußte den neuen DIN-Zeichensatz entwerfen. Dieser hat jetzt zwar deutsche Umlaute und und einige andere Zeichen zusätzlich erhalten, allerdings auf Kosten der Grafikzeichen, von denen der deutsche Zeichensatz weitaus weniger zur Verfügung hat als der amerikanische.
Deshalb ist es nicht erstaunlich, daß die Bildschirmcodes für Grafikzeichen im DIN-Modus mit den entsprechenden Codes im ASCII-Zeichensatz äußerst wenige Gemeinsamkeiten haben. Nur folgende Grafikzeichen haben in ASCII- und DIN-Modus denselben Bildschirmcode:

<SHIFT>+<L> (im Grafikzeichensatz), Bildschirmcode 76
<SHIFT>+<O> (im Grafikzeichensatz), Bildschirmcode 79
<SHIFT>+<P> (im Grafikzeichensatz), Bildschirmcode 80

Ansonsten aber ist keine Übereinstimmung vorhanden.
Deshalb empfiehlt es sich, bei der Verwendung von Grafikzeichen entweder ganz auf POKEs in den Bildschirmspeicher zu verzichten oder eine Umrechnungsformel zu gebrauchen, die zu einem ASCII-Code, den man ja über ASC(»Zeichen«) ohne großen Aufwand erhält, den entsprechenden Bildschirmcode berechnet. Eine solche Formel ist folgende (in »A« muß der ASCII-Code stehen, der Bildschirmcode kommt dann in die Variable »B«):

```
B=A+33*(A=255)+64*(A>63)+32*(A<96)-32*(A<160)+64*(A>191)
```

Die Funktionsweise dieses komplizierten Ausdrucks ist zwar auf den ersten Blick nicht verständlich, in 3.3 werden Sie aber alles darüber erfahren. Diese Formel funktioniert übrigens in beiden Zeichensätzen und sowohl auf dem C128 als auch auf dem C64.

3.1.2 ASCII-Codes

Gleich vorweg: eine Tabelle der ASCII-Codes finden Sie im Anhang des C128-Handbuches ab Seite A-7. Diese Tabelle ist so gut gelungen, daß es sich erübrigt, eine ähnliche Übersicht in dieses Buch aufzunehmen.
Hier beschäftigen wir uns nur mit dem Vergleich der ASCII-Codes.

ASCII-Codes von druckenden Zeichen

Falls Sie mit dem Begriff »druckende Zeichen« Schwierigkeiten haben: So nennt man die Zeichen mit den ASCII-Codes 32-127 und 1, da sie bei Ausgabe über PRINT direkt am Bildschirm erscheinen (Buchstaben, Zahlen, Grafikzeichen, Symbole etc.).
Die andere Gruppe trägt die Bezeichnung »Steuerzeichen«. Steuerzeichen rufen nur eine Steuerfunktion hervor (z.B. Beeinflussung der Schriftfarbe, Reversdruck, Umschaltung auf Grafik- oder Klein-/Groß-Zeichensatz, Zeilenvorschub usw.).
Die ASCII-Codes der druckenden Zeichen unterscheiden sich im amerikanischen Zeichensatz nicht von ihren C64-Äquivalenten. Der DIN-Zeichensatz hat die ASCII-Codes im ASCII-Zeichensatz nur teilweise beibehalten. Die neuen Codes können Sie in der besagten Tabelle im Handbuch finden (Spalte für den DIN-Modus).

ASCII-Codes von Steuerzeichen

Bei den Steuerzeichen ist eine ganze Reihe zusätzlicher Codes hinzugekommen; dies ist auch nötig, weil einige neuen Tasten einen eigenen ASCII-Code benötigen.
Alle C64-Steuerzeichen werden auch vom C128 verstanden, wenn man von einer Ausnahme absieht: Die Funktionen zum Blockieren und Entriegeln von <SHIFT>+<CBM> liegen jetzt nicht mehr auf den Codes 8 und 9, sondern auf 11 (Blockieren) und 12 (Entriegeln). Dies wurde übrigens schon bei der Erklärung der <CONTROL>-Funktionen in Kapitel 2 behandelt.
Folgende ASCII-Codes kennt der C128 als Steuerzeichen, die beim C64 nicht vorhanden sind (Tabelle 3.1):

Tabelle 3.1: *Zusätzliche Steuercodes des C128*

ASCII-Code	Wirkung
2 »b«	Unterstreichmodus ein (nur 80-Zeichen-Bildschirm)
7 »g«	akustisches Klingelzeichen
9 »i«	Tabulator (Code der <TAB>-Taste)
10 »j«	Zeilenvorschub (Code der <LINE FEED>-Taste)
15 »o«	Blinkmodus ein (nur 80-Zeichen-Bildschirm)
24 »x«	Tabulator setzen oder löschen (<SHIFT>+<TAB>)
27	Escape (Code der <ESC>-Taste, siehe 2.1.2)
130	Unterstreichmodus aus (nur 80-Zeichen-Bildschirm)
143	Blinkmodus aus (nur 80-Zeichen-Bildschirm)

Steht hinter einem Code in Anführungszeichen eine Taste, so ist dies die Taste, die zusammen mit <CONTROL> betätigt werden muß, um diese Funktion im Direkt-Eingabe-Modus zu simulieren (siehe auch Kapitel 2).
Noch ein kleiner Hinweis: die »Commdore-128-Codetabelle« im Handbuch, die so oft erwähnt wurde, ist an einer (einzigen) Stelle unvollständig. Tragen Sie deshalb bitte beim Code 29 (Seite A-7) in der Spalte »Hex« (hexadezimaler Zahlenwert) »1d« und unter »Funktion« den Text »CRSR rechts« ein. Nach dieser Ergänzung können Sie bedenkenlos af die Codetabelle zurückgreifen, die Ihnen sicher noch eine wertvolle Hilfe beim Programmieren sein wird.
Abschließend sei gesagt, daß sich im ASCII-Modus (<CAPS LOCK> entriegelt) wenig an den ASCII- und überhaupt nichts an den Bildschirmcodes gegenüber dem C64 ändert. Die erweiterten Funktionen des CHR$-Befehls aufgrund neuer ASCII-Codes von Steuerzeichen sind eine große Hilfe beim Programmieren. An den Beispielprogrammen dieses Buches werden Sie beispielsweise häufig sehen, daß die Codes für das Klingelzeichen und die ESC-Taste bald ähnlich geläufig sind wie die Farbsteuerzeichen, Revers ein (RVS ON) und aus (RVS OF) und Bildschirm löschen (CLR).

3.2 Erste Unterschiede der Basic-Interpreter

Das Basic 2.0 des C64, das durch akute Befehlsarmut »glänzt«, unterstützte folgende Bereiche überhaupt nicht oder nur unzureichend:
- Grafik
- Sound
- Strukturierte Programmierung
- Komfortable Programmierung (Programmierhilfen)
- Ein-/Ausgabe und dabei Diskettenprogrammierung im besonderen.

Die Folge war, daß eine mittlerweile unüberschaubare Anzahl von Basic-Erweiterungen auf den Markt kam, die Zusatzbefehle für die genannten Bereiche zur Verfügung stellen. Eingeleitet wurde diese Welle durch Simon's Basic, die bekannteste und verbreitetste C64-Basic-Erweiterung.
Doch die Entwickler des Basic 2.0 hatten ihrerseits auch weitergearbeitet. Basic 4.0 hieß eine Version, die zwar Diskettenbefehle hat, ansonsten aber mit dem Basic 2.0 identisch ist. Dieses Basic 4.0 wurde in größeren CBM-Computern eingebaut (CBM 80xx-Reihe).
Zum C16/116/Plus 4 wurde dann ein Basic entwickelt, das auch eine Weiterentwicklung des Basic 2.0 (nicht des Basic 4.0!) ist, um den C16/116/Plus 4 als Einsteigercomputer bedienungsfreundlich zu bedienungsfreundlich zu gestalten.
Das mächtige Basic 7.0, das unser C128 versteht, besitzt alle Befehle der Versionen 2.0, 3.5, 4.0 und einige neue Kommandos, z.B. zur Spriteprogrammierung (Basic 3.5 kennt zwar viele Grafikbefehle, aber keine Spritekommandos, da der C16/116/Plus 4 nicht in der Lage sind, Sprites zu erzeugen wie der C64/128).

Die Erweiterungen beschränkten sich jedoch nicht nur darauf, neue Befehle hinzuzufügen, sondern erfreulicherweise wurden auch alte Unzulänglichkeiten von Basic-2.0-Befehlen ausgebügelt. Die sich daraus ergebenden Änderungen werden in diesem Unterkapitel behandelt.

Länge der Eingabezeilen

Auch der C64 unterscheidet zwischen einer »echten« und einer »logischen« Eingabezeile: eine echte Eingabezeile beim C64 hat 40 Zeichen, ist also genau eine Bildschirmzeile (beim C128 abhängig von Bildschirmformat und Window-Größe). Weil aber 40 Zeichen recht eng bemessen ist, läßt der C64 bei INPUT oder im Basic-Eingabe-Modus auch Eingaben mit einer Länge von zwei echten Zeilen, also bis zu 80 Zeichen Länge, zu.
Beim C128 ist die maximale Eingabelänge mit 160 festgesetzt. Im 40-Zeichen-Modus sind das vier Bildschirmzeilen (160:40 = 4), im 80-Zeichen-Modus logischerweise zwei (160:80 = 2). Dies heißt, daß man also beim C128 ohne Probleme doppelt so lange Basic-Zeilen eingeben kann wie beim C64.
Auch wenn man durch die Window-Technik (siehe 2.1.3) die Anzahl der Zeichen pro Zeile einschränkt, kann dennoch eine Eingabe 160 Zeichen lang sein, sie beansprucht dann aber mehr Window-Zeilen.
Versucht man eine längere Eingabe, wird diese mit der Meldung »?STRING TOO LONG ERROR« zurückgewiesen.

Befehlsabkürzungen im C128-Modus

Da man die meisten Basic-Befehle nicht ausschreiben, sondern nur ein bis zwei Buchstaben und ein geshiftetes Zeichen angeben muß, ist die Eingabe dieser Befehlsabkürzungen dem C64-Anwender zur Gewohnheit geworden. Bei Benutzung von Abkürzungen für Basic-2.0-Befehlswörter im C128-Modus ergeben sich folgende Änderungen gegenüber dem C64:
CONT, END und SPC(können nicht mehr abgekürzt werden.
PEEK, POKE, READ und STOP werden jetzt durch Angabe der zwei ersten Zeichen ohne und des dritten Zeichens mit SHIFT eingegeben, also z.B. <P><E><SHIFT+E> für PEEK.
Wenn Sie dies mißachten, ergeben die bisherigen Abkürzungen folgende Basic-7.0-Befehle:

<C><SHIFT+O>	CONCAT	statt	CONT
<E><SHIFT+N>	ENVELOPE	statt	END
<P><SHIFT+E>	PEN	statt	PEEK
<P><SHIFT+O>	POT	statt	POKE
<R><SHIFT+E>	RENAME	statt	READ
<S><SHIFT+P>	SPRITE	statt	SPC<
<S><SHIFT+T>	STASH	statt	STOP

Leerstrings nach ASC

Der Basic-2.0-Befehl ASC, der in 3.1 noch einmal erklärt wurde, wandelt bekannterweise das erste Zeichens eines String-Ausdrucks in den dazugehörigen ASCII-Code um. Der C64 gibt

eine Fehlermeldung (»?ILLEGAL QUANTITY«) aus, wenn er folgenden Befehl abarbeiten soll:

```
PRINT ASC("")
```

Dies gilt auch für die leicht abgewandelte Form

```
A$="":PRINT ASC(A$)
```

Diese Fehleranfälligkeit ist vor allem ein Ärgernis, wenn ein Byte über GET# von Diskette geholt wird und in den korrekten ASCII-Code umgewandelt werden soll: wurde beim Schreiben der Datei CHR$(0) gesendet, faßt die ASC-Funktion des Basic 2.0 dies als Leerstring auf und erzeugt eine Fehlermeldung.
Mit der Hilfsformulierung

```
PRINT ASC(A$+CHR$(0))
```

kann man zwar diese kleine Unzulänglichkeit umgehen, aber eine von Natur aus korrekte Arbeitsweise des ASC-Befehls ist unbestritten die bessere Lösung. In Basic 7.0 ergibt

```
PRINT ASC("")
```

jetzt sinnvollerweise den Wert 0 und nicht, wie auf dem C64, eine Fehlermeldung. Der oben genannte Hilfsausdruck gehört somit der Vergangenheit an.

Systemvariablen

Zusätzlich zu den reservierten Systemvariablen ST, TI und TI$ von Basic 2.0 kennt Basic 7.0 folgende weitere, die ebenso für eigene Programme tabu sind: ER, EL, DS, DS$. Deren Bedeutung wird in 3.4.1-3.4.6 besprochen.

RESTORE mit Zeilennummer

Der RESTORE-Befehl wurde um die Möglichkeit erweitert, eine Zeilennummer anzugeben, auf die der READ/DATA-Zeiger gestellt werden soll. Eine berechnete Zeilennummer (etwa RESTORE A*5 oder RESTORE X) ist allerdings nicht möglich, sondern nur Zahlen (100, 500, 1000 etc.). Diese Einschränkung kennen Sie ja schon von GOTO, GOSUB und RUN.
Verzichtet man auf die Zeilennummernangabe, dann arbeitet RESTORE wie beim C64, d.h. der READ/DATA-Zeiger wird auf den Programmanfang gestellt.
Ein Beispiel für RESTORE mit Zeilennummer finden Sie übrigens im C128-Handbuch auf Seite 4-100, wo der RESTORE-Befehl besprochen wird.

MID$ links vom Zuweisungszeichen

In Basic 2.0 kann MID$ nur als Funktion verwendet werden.
Basic 7.0 erlaubt es, MID$ links vom Zuweisungszeichen »=« zu plazieren und so einen Teil eines Strings zu verändern. Beispiel:

```
A$="MAYER":REM String definieren
MID$(A$,2,2)="EI":REM Ab zweitem Zeichen zwei Zeichen ändern
PRINT A$:REM Veränderten String ausgeben
```

Ergebnis: MEIER (das »AY« wurde durch »EI« ersetzt).

Eine Schlüsselfunktion bekommt der »MID$(...)=«-Befehl bei einer Eingaberoutine, die wir in 3.6.4 entwickeln werden. Dort wird jede gedrückte Taste direkt über »MID$(...)=« in den Eingabestring aufgenommen.
Achtung: die Befehle LEFT$ und RIGHT$, die ähnlich wie MID$ arbeiten, können weiterhin nur als Funktionen verwendet werden. Es ist aber möglich, durch MID$-Konstruktionen auch die Befehle »LEFT$(...)=« und »RIGHT$(...)=« zu schaffen.
Um »LEFT$(A$,X)=B$« zu simulieren, schreibt man:

```
FOR I=1 TO X:MID$(A$,I,1)=MID$(B$,I,1):NEXT I
```

Bei RIGHT$ geht es ähnlich, allerdings muß die Stringlänge berücksichtigt werden.

```
RIGHT$(A$,X)=B$
```

wird folgendermaßen programmiert:

```
L=LEN(A$):FOR I= 1 TO X:MID$(A$,L-X+I,1)=MID$(B$,I,1):NEXT I
```

Programmiertes Listing

Der Befehl »LIST« eignet sich in Basic 2.0 nur im Direktmodus, da er nach seiner Ausführung nicht ins Programm zurückspringt, sondern in den Eingabemodus.
Beim C128-Basic ist dies verbessert worden, LIST führt jetzt nicht mehr zum Programmende. Mit einem kleinen Trick ist dies übrigens auch auf dem C64 möglich. Dieses raffinierte Verfahren sei hier nebenbei vorgestellt, wenn Sie sich aber nicht mehr für die Programmierung des C64 interessieren, können Sie die folgenden Ausführungen überlesen.
Als Beispiel soll folgendes Programm dienen:

```
100 LIST
110 PRINT "PROGRAMM WURDE FORTGESETZT"
```

In Basic 7.0 funktioniert dies, in Basic 2.0 jedoch nicht, da nach LIST abgebrochen wird und Zeile 110 nie zur Ausführung gelangt. Nach Ergänzung folgender Zeilen läuft das Programm auf dem C64 (auf dem C128 allerdings nicht, da die POKEs nur für den C64 gelten):

```
10 poke 198,6:REM Anzahl der Zeichen im Tastaturpuffer festlegen
20 poke 631,asc("g")
30 poke 632,asc("O"):REM <SHIFT>+<O>
40 poke 633,asc("1"):REM Zeilennummer »110«
50 poke 634,asc("1"):REM im ASCII-Code
60 poke 635,asc("0"):REM ablegen
70 poke 636,13:REM 13 = ASCII-Code von <RETURN>
```

Probieren Sie's im C64-Modus aus!

Die Zeilen 10-70, die ergänzt werden müssen, täuschen dem Computer nämlich vor, daß »GOTO 110« (in abgekürzter Schreibweise) eingegeben und RETURN gedrückt wird. Während des laufenden Programms interessiert sich der C128 nicht dafür, aber bei Programmende (nach dem Listen) kommen diese simulierten Tastendrücke zur Ausführung und veranlassen eine Fortsetzung des Programms bei Zeile 110.
Lassen Sie das Programm einmal laufen, dann werden Sie sehen, daß nach dem Listen die Meldung »READY.« erscheint (das Programm wird ja zunächst durch LIST beendet), darunter aber der GOTO-Befehl steht, der durch die POKE-Befehle in den Zeilen 10-70 vorbereitet wird.
In 3.7.5 erfahren Sie, wie man Tastendrücke auf dem C128 in Basic simuliert, und vor allem, welche interessanten Möglichkeiten sich daraus ergeben. Über die Programmierung dieses Tricks auf dem C64 will ich mich hier nicht länger ausbreiten, da dieses Buch kein Werk über die Programmierung des C64 ist. Ich kann Sie deshalb auf das »C64-Profihandbuch« von Markt & Technik, Nummer MT 749 verweisen, in dessen Tips-und-Tricks-Kapitel einige Anwendungen zu diesem Thema vorgestellt werden.
Zurück zum LIST-Befehl in Basic 7.0.
Bei der Ausgabe eines Basicprogramms auf den Drucker kann man nun statt zwei Befehlseingaben (»OPEN...:CMD...:LIST« <RETURN> und »PRINT#...:CLOSE...« <RETURN>) folgende Zeile eingeben:

```
OPEN4,4:CMD4:LIST:PRINT4:CLOSE4
```

Durch diese Zeile wird das etwas störende »READY.« am Ende des Listings nicht auf den Drucker ausgegeben, sondern unterdrückt. Ist das »READY.« aber als Schlußmarkierung erwünscht, ist dieselbe Befehlsfolge wie beim C64 zu verwenden.

Der neue SYS-Befehl

Die Syntax für den SYS-Befehl des C64 lautet

```
SYS Adresse
```

»Adresse« ist dabei eine Zahl von 0 bis 65535, die die Startadresse eines Maschinenprogramms ist, das angesprungen werden soll.
Das bekannteste Beispiel vom C64 ist »SYS 64738« (nicht im C128-Modus eingeben!), was zum Reset (Herstellen des Einschaltzustandes) führt.
Beim C128 können mit dem SYS-Befehl auch die Werte der Prozessorregister A (Akkumulator), X (X-Register), Y (Y-Register) und S (Statusbyte) übermittelt werden.
Dies geschieht, indem je nach Bedarf weitere Parameter angehängt werden:

```
       SYS Adresse,A,X,Y,S
oder:  SYS Adresse,A,X,Y
oder:  SYS Adresse,A,X
oder:  SYS Adresse,A
oder:  SYS Adresse
oder:  SYS Adresse ,,X
```

oder: SYS Adresse ,A,,Y
oder: SYS Adresse ,,,,S
usw.

Näheres dazu steht im Kapitel 4.2 (Maschinensprache), da die Syntax und Funktionsweise SYS-Befehls nur für Assemblerprogrammierer wichtig sind.

RUN mit erweiterten Möglichkeiten

Nicht nur RESTORE und SYS sind auf dem C128 erweitert worden, auch der RUN-Befehl, einer der elementarsten Basic-Befehle überhaupt, wurde um eine recht nützliche Funktion ergänzt.
Man kann zwar weiterhin in Basic 7.0 »RUN« oder »RUN 100« usw. eingeben, um ein Programm zu starten, hat aber auch die Möglichkeit, ein Programm von Diskette einlesen und automatisch starten zu lassen:

```
RUN "FILENAME"
```

Dieser Befehl würde auf dem C64 zur Meldung »?UNDEF'D STATEMENT« führen, auf dem C128 lädt er das Basicprogramm »FILENAME« von Diskette und startet es automatisch. Weitere Informationen über die Syntax erhalten Sie in 3.4.5.

Organisation des Basic-Speichers

Beim C64 ist der Basic-Speicher recht einfach organisiert: das Basic-Programm beginnt, solange man dies nicht mit POKE-Befehlen vorsätzlich ändert, bei Adresse 2049 ($0801) und endet spätestens bei 40959 ($9FFF). Variablen liegen im selben Speicherbereich wie das Programm, weshalb Programm und Variablen zusammen nur 38911 Bytes Speicher beanspruchen dürfen.
Da der Beginn der Variablen im Speicher dem Ende des Programms unmittelbar folgt (also direkt von der Programmlänge abhängig ist), liegen Programm und Variablen »an einem Stück« im Speicher.
Um den Aufbau des C128-Basic-Speichers zu erklären, müssen wir kurz auf die Speicherorganisation des C128 eingehen.
Der Arbeitsspeicher von 128K (also ohne Ausbau auf 256K oder 512K) ist, damit der C128 diese Speichermenge verwalten kann, in zwei »Speicherbanks« von je 64K Größe unterteilt (ein 8-Bit-Computer wie der C128 kann nur auf 64K gleichzeitig zugreifen). Zwischen diesen beiden Banks, die als »Bank 0« und »Bank 1« bezeichnet werden, wird bei Bedarf umgeschaltet, d.h. der Basic-7.0-Interpreter greift entweder auf das Programm oder auf die Variablen zu.
Da ein Teil jeder Bank für verschiedene Arbeitsbereiche aufgewendet werden muß, stehen nicht 128K, sondern »nur« 122365 Bytes (ca. 119.5K) für das Basic-Programm und seine Variablen zur Verfügung.
Der dann noch verbleibende Speicher in Bank 0 nimmt das Basic-Programm, der Restspeicher in Bank 1 die Basic-Variablen auf, so daß je ca. 60K für das Programm und die Variablen bleiben.

Allerdings ist es nicht möglich, Programm oder Variablen über mehr als eine Speicherbank zu verteilen; das Programm kann nur in Bank 0, die Variablen können nur in Bank 1 liegen. Sollte der C128 Bank 0 und Bank 1 als zusammenhängenden Speicherbereich verwalten, würde dies eine viel umständlichere Programmierung des Basic-Interpreters erfordern und wäre für die ohnehin recht niedrige Arbeitsgeschwindigkeit des Basic 7.0 in einem unvertretbarem Ausmaß abträglich.
Ein Vorteil, der sich aus der neuen Verwaltung des Basic-Speichers ergibt, ist bei der Programmentwicklung, insbesondere beim Debugging (Fehlersuche und -berichtigung), nicht zu unterschätzen.
Während beim C64 das Ändern einer Programmzeile automatisch einen CLR (Löschen der Variablen) auslöst, kann man in Basic 7.0 einen Programmfehler berichtigen, ohne daß die Variablen gelöscht werden; dies ermöglicht beispielsweise ein Fortsetzen des Programms ab der korrigierten Stelle mittels GOTO.
Dies funktioniert beim C128 nur, weil ein Ändern des in Bank 0 liegenden Programms die Variablen, die ja in Bank 1 liegen, nicht berührt; beim C64 hingegen belegen Programm und Variablen gemeinsam einen Speicherbereich.

FRE - Wieviel Speicher haben wir noch?

In Basic 2.0 kann die FRE-Funktion nur Aussagen über den vorhandenen Gesamtspeicher machen. Da die C128-Speicherverwaltung anders als beim C64 ist, wie wir soeben besprochen haben, wurde auch die FRE-Funktion angepaßt. Jetzt ist das Argument von FRE nicht mehr ein Dummy-Wert (Wert, der zwar angegeben werden muß, aber keine Auswirkung auf die Funktion hat), sondern bestimmt, ob der freie Speicher für das Basic-Programm oder derjenige für die Basic-Variablen genannt werden soll:

FRE (0) = freier Speicher in Bank 0 = freier Programmspeicher
FRE (1) = freier Speicher in Bank 1 = freier Variablenspeicher

Geschwindigkeitsvergleich zwischen Basic 2.0 und Basic 7.0

Wie wir gesehen haben, muß der C128 zwischen verschiedenen Banks umschalten. Dieses »Bank-Switching« (so lautet der Fachausdruck) bringt aber eine Verlangsamung des Basic-Interpreters mit sich, da er nicht direkt auf den Speicher zugreifen kann, sondern nur über spezielle Umschalt-Routinen, die eine deutlich längere Verarbeitungszeit als einzelne Schreib-/Lese-Befehle benötigen.
Ein weiterer Grund, warum das Basic 7.0 des C128 langsamer als das Basic 2.0 des C64 ist, ist die größere Anzahl von Befehlen, die dekodiert werden, weshalb mehr Befehlscodes (»Tokens«) existieren müssen.
Um die Relationen zu sehen, können Sie folgende Zeile im Direktmodus laufen lassen, die nur aus Basic-2.0-Befehlen besteht und infolgedessen sowohl auf dem C64 als auch auf dem C128 läuft:

```
CLR:TI$="000000":FORI=1TO1000:NEXT:PRINTTI
```

Diese Zeile druckt eine Zahl aus, die angibt, wieviel 1/60 Sekunden die FOR-NEXT-Leerschleife mit 1000 Durchläufen gedauert hat.
Im Einschaltzustand des C128, also ohne vorheriges »Tuning« mit dem Basic-7.0-Befehl FAST, wird »86« gemeldet, also fast 1.5 Sekunden Arbeitsdauer. Das Basic 2.0 im C64-Modus kommt nur auf »62«, also ziemlich genau 1 Sekunde. Der alte VC 20 wäre sogar noch etwas schneller, was wir aber hier vernachlässigen wollen.
Beschleunigt man jedoch den C128 mit dem FAST-Befehl (siehe 2.2.4), was eine Zeitersparnis von über 50 Prozent bringt, erhält man den Wert »40«, also weniger als 0.75 Sekunden. Dann ist der C128 endlich schneller als der C64, doch die eingeschränkte Verwendbarkeit des FAST-Modus (siehe 2.2.4) darf man nicht vergessen.
Abschließend ist zu sagen, daß sich der C128 seinen großen Arbeitsspeicher und die ungeheuere Befehlsvielfalt durch eine etwas verringerte Arbeitsgeschwindigkeit gegenüber dem C64 erkauft, da der C64 und der C128 von der Hardwareseite aus gleich schnell wären. Dennoch kann der Benchmark-Test (Geschwindigkeitstest) mit der FOR-NEXT-Schleife täuschen, denn bei Einsatz von komfortablen Basic-7.0-Befehlen anstatt von langsamen Basic-2.0-Routinen lassen sich in den meisten Bereichen erhebliche Beschleunigen gegenüber dem C64 erzielen (Grafik, Sound, Stringverarbeitung, Ein-/Ausgabe, Schleifen).

3.3 Was nicht im C64-Handbuch steht ...

Eine sehr nützliche Eigenschaft des Basic 2.0, die auch das Basic 7.0 hat, wird im C64-Handbuch verschwiegen und im C128-Handbuch nur unzureichend erklärt: die numerische Auswertung von Bedingungen.
Damit ist gemeint, daß man IF-ähnliche Abfragen in mathematische Formeln einbinden kann, um IF...THEN-Konstruktionen zu umgehen.
Zur Demonstrationgeben Sie bitte folgende Befehle im Direktmodus ein, ein, auch wenn man von diesen erwarten könnte, daß ein »SYNTAX ERROR« entsteht:

```
PRINT 1<2     Antwort des Computers: -1
PRINT 1=2     Antwort des Computers:  0
PRINT 1>2"    Antwort des Computers:  0
PRINT 1<>2    Antwort des Computers: -1
```

Man erkennt bei genauerer Betrachtung, daß der Computer dann »-1« ausgibt, wenn der hinter PRINT stehende Ausdruck wahr ist (wie etwa »1<2«; die Antwort ist »0«, wenn der Ausdruck falsch ist (wie etwa »1=2«). Offensichtlich wertet der Computer diese mathematischen Aussagen aus, denn das Basic errechnet für einen logischen Ausdruck (erkennbar an einem Gleichheits- oder Ungleichheitszeichen) einen Zahlenwert, der den Wahrheitswert (»wahr« oder »falsch«) angibt.

Setzt man einen logischen Ausdruck in Klammern, kann man sogar mit der zahlenmäßigen Bewertung (0 oder -1) weiterrechnen:

```
PRINT 5*(1<2)
```

ergibt »-5«, da (1<2) wahr ist und somit den Wert (-1) hat, der mit 5 multipliziert (-5) ergibt.

```
PRINT 5*(1>2)
```

ergibt »0«, da (1>2) falsch ist und somit den Wert 0 hat, der mit 5 multipliziert 0 ergibt.
Wie man diese Eigenheit des Basic 2.0, die auch im Basic 7.0 enthalten ist, gewinnbringend einsetzt, zeigt dieser Abschnitt auf.
Dabei kommt es nicht so sehr darauf an, darauf an, zu verstehen, warum dieser oder jener Trick funktioniert (dazu muß man einiges an mathematischem Verständnis aufbringen), sondern vielmehr darauf, daß Sie einen Einblick bekommen, wie man dies in der Praxis verwendet. Sollten Sie also eine mathematische Beweisführung nicht verstehen, können Sie diese ohne Bedenken überfliegen.

IF-Abfragen vorbereiten

Sind Sie schon bei der Analyse eines Basic-Programms über eine Anweisung wie »IF A THEN ...« gestolpert und haben sich gewundert, warum der IF-Befehl eine Variable (A) anstelle eines logischen Ausdrucks (wie z.B. A>5) abfragt? Dann erfahren Sie jetzt, warum diese auf den ersten Blick unsinnige Anweisung durchaus ihre Berechtigung hat.
Betrachtet man die Funktionsweise des IF-Befehls, arbeitet er nach folgendem vereinfachten Schema:

a) Zahlenwert (!) holen

Wie wir soeben gesehen haben, werden Vergleichsoperationen mit »=«, »<«, »>« usw. automatisch ins Zahlenformat (0 oder -1) umgewandelt. Deshalb muß der IF-Befehl nicht selbst prüfen, ob die ihm folgende Bedingung wahr ist (wie man es eigentlich erwarten würde), sondern nur die Routine aufrufen, die einen Zahlen-Ausdruck auswertet, d.h. der nach IF stehende Ausdruck wird nicht anders behandelt als eine Zahlenangabe nach einer Variablenzuweisung (!).

b) Zahlenwert weiterverarbeiten

Ist der Wert 0, so wird die Bearbeitung nicht nach THEN fortgesetzt (beim C128 wird unter Umständen die ELSE-Behandlung durchgeführt). Ist der Wert nicht 0, sondern z.B. -1, wird der nächste Befehl nach »THEN« angesprungen.
Jetzt ist Ihnen sicher klar, weshalb folgender IF-Befehl immer die THEN-Behandlung auslösen würde:

```
IF -1 THEN ...
```

Dies gilt auch für

```
A=-1:IF A THEN ...
```

Entsprechend würde folgender IF-Befehl nie in den THEN-Teil springen, weil 0 für eine nicht erfüllte IF-Bedingung steht:

```
IF 0 THEN ...
```

oder

```
A=0:IF A THEN ...
```

An diesen (unsinnigen) Beispielen kann man sehen, daß es möglich ist, den Wahrheitswert einer Aussage zuerst zu bestimmen, in einer Variablen zu speichern und erst im Bedarfsfall mit IF weiterzuverarbeiten. Etwas umfangreicher ist folgendes Beispielprogramm, das Sie bitte eingeben (da Sie aus dem Abtippen lernen sollen, befindet sich dieses Programm nicht auf der Programmdiskette):

```
10 INPUT "BITTE ZWEI ZAHLEN:";A,B
20 V=(A=B):REM V enthält Vergleichsergebnis (0 oder -1)
30 PRINT "VERGLEICHSERGEBNIS:";V
40 PRINT "DIE ZWEI EINGEGEBENEN ZAHLEN SIND:"
50 IF V THEN PRINT "GLEICH"
60 IF NOT V THEN PRINT "VERSCHIEDEN"
```

Das Programm fordert Sie zur Eingabe von zwei Zahlen auf und sagt Ihnen dann, ob diese gleich oder verschieden sind. Dabei wird nicht in einer IF-Abfrage geprüft, ob A und B gleich sind, sondern unmittelbar nach der Eingabe der Vergleich durchgeführt (Zeile 20). Das Ergebnis (gleich/verschieden) kommt in die Variable V, die als »Vergleichsergebnis« ausgegeben wird (Zeile 30), um das Verständnis zu erleichtern.
Die beiden IF-Befehle in den Zeilen 50/60 prüfen dann nur noch den Wert der Variablen V. Der große Vorteil davon ist, daß die Variable V das Vergleichsergebnis über eine beliebig lange Zeit erhält und eine IF-Verarbeitung auch später erfolgen kann.
Folgendes Programm listet sich bei Eingabe von »J« selbst:

```
10 INPUT "LISTEN (J/N)";A$
20 V=(A$="J"):REM A$ mit "J" vergleichen, Ergebnis in Variable V
30 A$="X":REM A$ neubelegen
40 IF V THEN LIST:REM V abfragen
```

Der Fachausdruck für die Variable V in diesem Programm lautet übrigens »Flag« (»Flagge«). Mit Flag werden Variablen oder Speicherplätze bezeichnet, die einem Programm als Information dienen, ob eine bestimmte Funktion ausgelöst werden soll oder nicht. In unserem Fall ist V ein List-Flag, da der Inhalt von V darüber entscheidet, ob das Programm gelistet wird (V=-1) oder nicht (V=0).
Auch das Betriebssystem und der Basic-Interpreter haben Flags; so sagt beispielsweise die Adresse 215 aus, ob sich der C128 im 40- oder 80-Zeichen-Modus befindet (siehe 2.2.2).
Die Bedeutung des Begriffs »Flag« sollten Sie sich merken, da in einigen Programmdokumentationen, die noch folgen werden, die Kenntnis dieses Ausdrucks vorausgesetzt werden wird.

Addition und Subtraktion mit Ober- und Untergrenze

Will man eine Variable X um 10 erhöhen, schreibt man »X=X+10«. Soll dabei X nie den Wert 100 erreichen, muß X vor der Addition von 10 kleiner als 90 sein. Dies ist mit Hilfe von IF...THEN leicht formuliert:

```
IF X<90 THEN X=X+10
```

Soweit nichts Neues.
Es besteht aber die Möglichkeit, mit einem einzigen Befehl dasselbe Ziel ohne IF...THEN zu erreichen:

```
X=X-10*(X<90) entspricht "IF X<90 THEN X=X+10"
```

Dies ist zunächst etwas unübersichtlich. Vor allem könnte der Operator »-« stören, wenn man bedenkt, daß im Endeffekt eine Addition durchgeführt werden soll.
Ein mathematischer Beweis der Richtigkeit dieses Ausdrucks ist erforderlich und dem Verständnis dienlich.
Beweis:

```
"X=X-10*(X<90)" entspricht "IF X<90 THEN X=X+10"
```

Während »X=X+10« immer gleich arbeitet, müssen hier zwei unterschiedliche Behandlungen betrachtet werden.

Fall 1: (X<90) ist wahr. Bei der IF-Lösung wird 10 addiert.

Wir können für diesen Spezialfall den Klammerausdruck durch -1 ersetzen, da dies der Basic-Interpreter ohnehin tut:

```
X=X-10*(-1)
```

Dies können wir ausmultiplizieren in:

```
X=X-(-10)
```

Bei Auflösung der Klammern ergibt sich:

```
X=X+10
```

Folgerung: Ist X<90, wirkt »X=X-10*(X<90)« wie »IF X<90 THEN X=X+10«!
Nun müssen wir noch nachweisen, daß dies auch für den Fall gilt, daß die Variable X nicht kleiner als 90 ist.

Fall 2: (X<90) ist falsch. Bei der IF-Lösung ändert sich X nicht.

In diesem Fall wird der Klammerausdruck vom Basic-Interpreter als »0« behandelt:

```
X=X-10*(0)
```

Rechnet man weiter, ergibt sich

```
X=X-0 oder: X=X (X wird nicht verändert).
```

Da die Subtraktion von 0 keinen Einfluß auf das Ergebnis hat, wird X – wie bei der IF-Lösung – nicht beeinflußt. Folglich arbeitet »X=X-10*(X<90)« auch dann wie »IF X<90 THEN X=X+10«, wenn X nicht kleiner als 90 ist, also der Ausdruck »X<90« nicht zutrifft.
Folgendes Demoprogramm soll dies veranschaulichen:

```
10 X=0
20 X=X-10*(X<90):REM entspricht: IF X<90 THEN X=X+10
30 PRINT X
40 FOR I=1 TO 250:NEXT:REM kleine Verzögerung
50 GOTO 20
```

Wie man sieht, kann man eine IF...THEN-Anweisung auf diese trickreiche Weise sparen. Es soll aber nicht verschwiegen werden, daß dies durch eine geringere Übersichtlichkeit erkauft wird.
Es sei noch kurz eine ähnliche Anwendung für die Subtraktion gezeigt. Diesmal soll eine Zahl von 100 in 10er-Schritten verringert werden, darf aber vor der Subtraktion nie größer als 10 sein, um zu gewährleisten, daß das Subtraktionsergebnis immer positiv ist:

```
10 X=100
20 X=X+10*(X>10):REM entspricht: IF X>10 THEN X=X-10
30 PRINT X
40 FOR I=1 TO 250:NEXT
50 GOTO 20
```

In der Formel in Zeile 20 haben sich jetzt der Operator (»–« wurde zu »+») und die Bedingung (X>10 statt X<90) geändert, beide Lösung basieren aber auf dem gleichen Prinzip, das ausführlich erklärt wurde.
Zugegeben, diese Programmiertechnik ist sehr kompliziert und kann leider nur auf dem mathematischen Weg erläutert werden. Aber es spielt, wie schon gesagt, keine Rolle, ob Sie den mathematischen Beweis verstanden haben. Wichtig ist nur, daß Sie einen Einblick in diese trickreiche Programmierung bekommen haben. In späteren Kapiteln wird dies noch einmal aufgegriffen und an bestimmten Spezialfällen so so ausführlich zerpflückt, daß bestimmt keine Verständnisschwierigkeiten auftreten. Zum Abschluß von 3.3 soll Ihnen aber noch eine kleine Hilfe bei der Basic-Programmierung gegeben werden.

Umrechnung von ASCII- und Bildschirmcode

Unglücklicherweise unterscheiden sich die ASCII-Codes der Zeichen erheblich von den korrespondierenden Bildschirmcodes, weshalb sehr häufig Umwandlungen von einem Format ins andere anfallen.
Meist sind solche Umwandlungsroutinen mehrere Zeilen lang, da eine Reihe von IF...THEN-Verzweigungen anfällt. Es geht aber auch viel einfacher, wenn man folgende Zeilen an den Anfang eines Programms setzt:

```
10 DEF FN AB(A)=A+33*(A=255)+64*(A>63)+32*(A<96)-32*(A<160)+
64*(A>191)
20 DEF FN BA(B)=B+64+64*(B<64 AND B>31)+32*(B<96 AND B>63)
```

Dann stehen Ihnen die Funktionen AB (Ascii- in Bildschirmcode) und BA (Bildschirm- in ASCII-Code zur Verfügung):
FN AB(x) ergibt den Bildschirmcode eines Zeichens mit dem ASCII-Code x
FN AB(ASC(«X»)) ergibt den Bildschirmcode des Zeichens X
FN BA(x) ergibt den ASCII-Code eines Zeichens mit dem Bildschirmcode x

Achtung: Fehlerhafte Angaben werden in der Regel nicht aussortiert, sondern bringen auch ein fehlerhaftes Ergebnis!
Die beiden Umrechnungsfunktionen werden im Programm MATRIX-VERGLEICH» (Listing-Nr. 2, siehe 2.3.2) in den Zeilen 260/270 definiert und dann ausgiebig benutzt.
Eine Erklärung der Funktionsweise ist nicht nötig, da diese beiden Formeln nur zur Erleichterung der Programmierung dienen sollen. Es sei aber beiläufig erwähnt, daß sogar mit NOT, AND und OR verknüpfte Aussagen ins Zahlenformat (0 oder -1) gewandelt werden, wie die Funktionsdefinition zu BA zeigt.

Bildschirmausgabe für 40- und 80-Zeichen-Modus

Die Einschaltmeldung des C128 wird immer zentriert ausgegeben, ob der Start in den 40- oder 80-Zeichen-Modus erfolgt. Dabei bedient sich das Betriebssystem eines simplen Tricks: Der Text ist normalerweise für den 40-Zeichen-Modus vorgesehen, im 80-Zeichen-Modus wird er um 20 Spalten eingerückt (80-40=40, 40:2=20).
Dies erreicht man in Basic mit folgendem Trick:

```
PRINT TAB(-20*(RWINDOW(2)=80));"TEXT"
```

Dann wird im 80-Zeichen-Modus die Ausgabe 20 Spalten später als im 40-Zeichen-Modus begonnen. Eine Zentrierung ist dabei nicht mitinbegriffen, kann aber über PRINT USING (siehe 3.4.5) programmiert werden.

3.4 Die neuen Befehle

Dieses Kapitel ist eine Einführung in diejenigen Befehle des Basic 7.0, die Sie noch nicht vom Basic 2.0 kennen.
Da diese Befehle ohnehin im C128-Handbuch erklärt werden, ist dieser Abschnitt keine Handbuch-Konkurrenz, sondern geht anders vor.
Die neuen Befehle werden nach Sinngruppen geordnet vorgestellt.
Die exakte Befehlsbeschreibung mit allen Syntax-Einzelheiten können Sie einer alphabetisch geordneten, sehr guten Übersicht im C128-Handbuch, das Sie bitte bereitlegen, entnehmen. Viele Befehle werden in diesem Buch nicht erklärt, wenn das Handbuch eine gute Beschreibung enthält; dann finden Sie selbstverständlich an der entsprechenden Stelle einen Verweis auf ein entsprechendes Kapitel im Handbuch.

Dieses Kapitel 3.4 verschafft nur einen Überblick über die neuen Befehle, damit Sie in Basic 7.0 bald Fuß fassen, vermittelt Tips & Tricks, gibt Anwendungsbeispiele und stellt Fehler oder Unzulänglichkeiten im Handbuch richtig.
Das Handbuch soll aber, wie gesagt, nicht ersetzt werden. Zum Einsatz der Befehlsübersicht im C128-Handbuch lesen Sie bitte dort das Kapitel 4.5 durch (Seiten 4-11/4-12); die alphabetische Befehlsübersicht der C128-spezifischen Basic-Kommandos finden Sie auf den Seiten 4-13 bis 4-134, eine Zusammenfassung der gemeinsamen Befehle von C64 und C128, also der Basic-2.0-Befehle, ab Seite 5-2.
Sie können die Handbucherklärungen schon vorher lesen, erforderlich ist dies aber nicht; vorausgesetzt wird nur, daß Sie bereits die Befehle des Basic 2.0 (C64/VC20) kennen.

Der Umgang mit den Abschnitten 3.4.1-3.4.7

Die Abschnitte 3.4.1-3.4.6 beschreiben jeweils eine Gruppe von Befehlen. Taucht ein neuer Befehl auf, so schlagen Sie bitte im Handbuch nach, wo eine genaue Befehlsbeschreibung steht; die Abschnitte 3.4.1-3.4.6 dienen als Wegweiser, 3.4.7 enthält dann abschließend eine Befehlsübersicht mit Kurzbeschreibung.
Damit Sie sich besser zurechtfinden, ist hier aufgeführt, welche Befehle in einem bestimmten Abschnitt erklärt werden.

Programmierhilfen, Abschnitt 3.4.1:

KEY, HELP, AUTO, RENUMBER, DELETE, TRON, TROFF, MONITOR

Strukturierte Programmierung, Abschnitt 3.4.2:

ELSE, BEGIN, BEND, DO, LOOP, EXIT, UNTIL, WHILE

Grafik, Abschnitt 3.4.3:

GRAPHIC, RGR, COLOR, RCLR, DRAW, RDOT, LOCATE, CIRCLE, BOX, PAINT,
CHAR, SCALE, WIDTH, SCNCLR,
GSHAPE, SSHAPE,
SPRDEF, SPRITE, RSPRITE, SPRCOLOR, RSPCOLOR, RSPPOS, SPRSAV, MOVSPR,
COLLISION, BUMP

Sound, Abschnitt 3.4.4:

SOUND, VOL, FILTER, PLAY, TEMPO, ENVELOPE

Ein-/Ausgabe, Abschnitt 3.4.5:

KEY, CHAR, COLOR, SCNCLR, GRAPHIC,
GETKEY, JOY, POT, PEN,
PRINT USING, PUDEF,
WINDOW, RWINDOW,

DLOAD, DSAVE, DVERIFY, BLOAD, BSAVE, BOOT
DIRECTORY, CATALOG, HEADER, SCRATCH, COLLECT, COPY, CONCAT,
RENAME, BACKUP, DS, DS$,
DOPEN, DCLOSE, DCLEAR, RECORD

Fehlerbehandlung und sonstige, Abschnitt 3.4.6:

TRAP, RESUME, ERR$, ER, EL, HELP,
DEC, HEX$, INSTR,
GO64,
BANK, STASH, FETCH, SWAP, QUIT, OFF,
SLEEP, RREG, FAST, SLOW, XOR, POINTER

3.4.1 Befehlsgruppe Programmierhilfen»

Von einem komfortablen Basic erwartet man, daß es schon die Programmentwicklung unterstützt. Einige Basic-7.0-Befehle haben nur diesen Zweck und sind deshalb meist nur auf der Befehlsebene (Direktmodus) verwendbar. Wird ein Befehl, der als Programmierhilfe vorgesehen ist, in einem Programm eingesetzt, tritt - von einigen wenigen Kommandos abgesehen - die Meldung »DIRECT MODE ONLY« auf.

KEY - Die programmierbare Funktionstastenbelegung

Hat man bei der Programmeingabe bestimmte Zeichenketten (z.B. Basic-Befehle) immer wieder einzutippen, kann man sich die Eingabe leichter machen, indem man diese Zeichenketten unter Zuhilfenahme des KEY-Befehls auf eine Funktionstaste legt, damit man sie später mit einem einzigen Tastendruck abrufen kann.
Der KEY-Befehl wurde schon in 2.1.1 erklärt, Sie können aber auch im C128-Handbuch auf Seite 4-77 nachschlagen, wenn Sie eine kurze tabellarische Zusammenfassung wünschen.
KEY kann auch in Programmen bedenkenlos verwendet werden.

HELP - Hilferuf an den Computer

Dieser Befehl wird bei Betätigen der <HELP>-Taste ausgelöst, kann aber natürlich ebenso zeichenweise eingetippt werden.
HELP listet die Zeile, in der der letzte Programmfehler diagnostiziert wurde, am Bildschirm (Fehler im Direktmodus können nicht mit HELP analysiert werden). Dabei wird ab der frühestmöglichen Fehlerquelle im 40-Zeichen-Modus revers, im 80-Zeichen-Modus unterstrichen gedruckt.
HELP wird auch im Programm-Modus ausgeführt.
Ein kleiner Tip: die Nummer der Zeile, in der ein Fehler aufgetreten ist, erfährt man über

```
PRINT EL
```

Die letzte aufgetretene Fehlermeldung bringt folgender Befehl auf den Bildschirm:

```
PRINT ERR$(ER)
```

Übrigens: logische Fehler, d.h. Fehler im Denkgerüst des Programms, können natürlich von HELP nicht ausfindig gemacht werden. Dies wäre von einem Computer zuviel verlangt!

AUTO – Automatik bei der Zeilennummer

Wenn man ein Programm fortlaufend eintippt, muß jedesmal die Zeilennummer neu eingegeben werden. Dies kann man sich ersparen, indem man dem Computer die gewünschte Schrittweite bei der Zeilennumerierung mitteilt:

```
AUTO 10
```

bewirkt auf den ersten Blick gar nichts. Gibt man aber eine Programmzeile (z.B. mit der Zeilennummer 100) ein, wird automatisch die nächste Zeilennummer (im Beispiel: 110) ausgegeben und der Cursor hinter dieser positioniert. Sie können aber den Cursor nach Belieben am Bildschirm bewegen und andere Zeilen editieren, was den AUTO-Befehl des C128 von seinen gleichnamigen Konkurrenten in manchen C64-Basic-Erweiterungen positiv abhebt.
Will man die automatische Zeilennumerierung ausschalten, drückt man zuerst auf <SHIFT>+ <CLR/HOME>, um den Bildschirm zu löschen, und schreibt dann:

```
AUTO        (oder: AUTO 0)
```

Dies schaltet die automatische Zeilennumerierung ab.
Der AUTO-Befehl ist gut gegen Fehlbedienung abgesichert: bei Eingabe einer Schrittweite, die größer als 63999 (höchste erlaubte Nummer einer Programmzeile) ist, wird ein SYNTAX ERROR» erzeugt; würde das Addieren der Schrittweite zur letzten Zeilennummer ein unerlaubtes Ergebnis bringen (>63999), so schaltet sich AUTO selbsttätig ab.

RENUMBER – Neunumerierung eines Programms

Der Befehl »RENUMBER« ersetzt das umständliche Umnumerieren eines Programms von Hand. Mit

```
RENUMBER
```

wird das im Speicher befindliche Programm in 10er-Schritten umnumeriert, wobei die erste Zeilennummer des neunumerierten Programms 10 ist. Es werden alle Zeilenangaben innerhalb eines Programms angepaßt, also auch die Zeilennummern hinter folgenden Befehlen:

GOTO (auch ON...GOTO)
GOSUB (auch ON...GOSUB)
THEN
ELSE
RESTORE
RESUME
TRAP
COLLISION

Laut Handbuch werden auch Referenzen nach EL (siehe 3.4.6) von der Neunumerierung betroffen, allerdings hat sich dies im Test und bei Analyse der RENUMBER-Routine des Basic-Interpreters nicht bewahrheitet. EL-Zeilennummern müssen also vom Programmierer selbst angepaßt werden.
Es ist auch möglich, dem RENUMBER-Befehl weitere Parameter zu übergeben. So wird bei

```
RENUMBER 100
```

das Programm in 10er-Schritten umnumeriert, aber die Startzeilennummer des Programms nach dem Neunumerieren ist nun 100.
Bei

```
RENUMBER 100,50
```

wird das Programm so umnumeriert, daß die Anfangszeile 100 ist. Zusätzlich ist die Schrittweite mit 50 festgelegt.
Soll nur ein Teil eines Programms »renumbert« werden, ist die erste Zeilennummer, ab der die Neunumerierung beginnen soll, mit anzugeben:

```
RENUMBER 100,50,2000
```

numeriert das Programm ab Zeile 2000 neu; Zeile 2000 wird zu Zeile 100, danach wird in 50er-Schritten numeriert.
Achtung: Wird nur ein Teilbereich neunumeriert, dürfen die Zeilennummern der Neunumerierung nicht mit Zeilennummern übereinstimmen, die bereits vorhanden sind; dies führt zur Meldung »?SYNTAX ERROR«.

Folgende Fehlermeldungen können bei RENUMBER auftreten:

DIRECT MODE ONLY:	RENUMBER läuft nur im Direktmodus.
ILLEGAL QUANTITY:	Parameter sind zu groß gewählt.
LINE NUMBER TOO LARGE:	Bei der Neunumerierung dürfen keine Zeilennummern entstehen, die 63999 überschreiten. (Diese Meldung steht nicht im Handbuch!)
UNRESOLVED REFERENCE:	Eine Zeilenreferenz im Programm springt eine nicht vorhandene Zeile an. (Diese Meldung steht nicht im Handbuch!)
SYNTAX ERROR:	- Fehler in Schreibweise - bestehende Zeilen müßten überschrieben werden (s.o.)

DELETE - Automatik beim Zeilenlöschen

Will man eine Zeile löschen, tippt man die Zeilennummer und drückt <RETURN>. Bei einer großen Anzahl von Zeilen ist dies aber sehr umständlich, solange man nicht das Zeilenlöschen automatisiert:

```
DELETE 100-200
```

löscht die Zeilen 100 bis 200 des im Speicher befindlichen Programms. Die Parameter von »DELETE« werden wie bei »LIST« angegeben.

```
DELETE 100
```

löscht nur Zeile 100.

```
DELETE -200
```

löscht alle Zeilen vom Programmanfang bis Zeile 200 (einschließlich).

```
DELETE 100-
```

löscht alle Zeilen ab Zeile 200 (einschließlich) bis zum Programmende.
Achtung: Beim Löschen von Zeilen mit DELETE besteht im Gegensatz zu NEW keine Möglichkeit, das Programm wiederherzustellen.

TRON und TROFF – Die Überwachungskamera für den Programmablauf

Der Befehl »TRON« (keine Parameter) schaltet die Ablaufüberwachung (Trace-Modus) ein. In diesem Zustand wird bei Ausführung eines Basic-Programms die Zeilennummer jeder durchlaufenen Anweisung (in eckigen Klammern) ausgegeben. Dies kann bei der Fehlersuche (Debugging) nützlich sein.
Die Ablaufüberwachung wird mit »TROFF« wieder ausgeschaltet.
Beide Befehle dürfen auch in einem Programm stehen.
In Simon's Basic geschieht die Ablaufverfolgung über den Befehl »TRACE«, der wesentlich besser als »TRON« und »TROFF« des Basic 7.0 arbeitet, da die Anzeige rechts oben am Bildschirm erfolgt (und nicht an beliebiger Position).

MONITOR – Maschinenprogramme analysieren und verändern

Mit dem Befehl MONITOR, der auch durch Drücken von <F8> ausgelöst wird, ruft man das integrierte Monitorprogramm des C128 auf. Da dieses in den Bereich der Maschinensprache gehört, wird es im Rahmen des Kapitels 4 (Maschinensprache), genauer gesagt im Abschnitt 4.1.1 erläutert. Eine Erklärung in diesem Zusammenhang würde die Basic-Programmierer unter Ihnen nur unnötig verwirren, da diese das Kommando MONITOR nicht verwenden können.

3.4.2 Befehlsgruppe »Strukturierte Programmierung«

Viele Basic-2.0-Programme kann man mit dem Begriff »Spaghetti-Code« umschreiben: vor lauter GOTOs und GOSUBs verliert man leicht den Überblick. Der C128 hat einige Befehle erhalten, die Struktur in Ihre Programme bringen, weil sie die Befehle »GOTO« und »GOSUB« vermeiden helfen.

ELSE - Alternative für den THEN-Befehl

ELSE (engl.: ansonsten) ist im Prinzip eine Alternative zum THEN-Befehl und den Simon's-Basic-Freunden keine Neuheit.
Ist die IF-Bedingung erfüllt, wird bekanntlich die Bearbeitung bei THEN fortgesetzt. In Basic 7.0 kann man auch hinter dem THEN-Teil, abgegrenzt durch einen Doppelpunkt, den Befehl ELSE angeben. Dann gibt es bei IF zwei Möglichkeiten:
- Bedingung ist erfüllt: THEN-Behandlung
- Bedingung nicht erfüllt: ELSE-Behandlung

Folgender Einzeiler mag als Beispiel dienen:

```
10 INPUT "ZAHL";Z:IF Z<0 THEN PRINT "NEGATIVE ZAHL":ELSE PRINT "POSITIVE
ZAHL ODER NULL"
```

Da nach ELSE auch ein weiterer IF-Befehl folgen kann, ist folgende Ergänzung zulässig:

```
10 INPUT "ZAHL";Z:IF Z<0 THEN PRINT "NEGATIVE ZAHL":ELSE IF Z=0 THEN
PRINT "NULL":ELSE PRINT "POSITIVE ZAHL"
```

Allerdings sollte man vorsichtig mit dem ELSE-Befehl umgehen, wenn mehrere IF...THEN-Konstruktionen verbunden werden, da es sonst unerwünschte Fehlfunktionen geben kann.
Folgende Regel gilt:
Stehen in einer Programmzeile mehrere IF...THEN-Unterscheidungen, von denen eine zusätzlich zum THEN-Teil ein »ELSE« hat, so müssen auch die vorhergehenden IF...THENs in dieser Zeile über eine ELSE-Behandlung verfügen, da sonst der C128 nicht ermitteln kann, auf welches IF...THEN sich ein bestimmtes ELSE bezieht.
Sie können selbst nachprüfen, daß diese Regel auf den obigen Einzeiler zutrifft. Wenn also ein IF...THEN...ELSE nicht korrekt zu arbeiten scheint, dann ist diese Regel zu prüfen.
Noch ein Hinweis: So wie man unmittelbar nach »THEN« den GOTO-Befehl weglassen kann (»IF ... THEN 50«), ist dies auch nach »ELSE« möglich.

BEGIN und BEND - Ergänzungen zum IF-Befehl

Da die Länge einer Basic-Zeile auch beim C128 stark begrenzt ist, hat man Schwierigkeiten, wenn man hinter THEN oder ELSE viele Anweisungen schreiben will; in der Regel muß man den GOTO-Befehl einsetzen, der zur Unübersichtlichkeit des Programms führt.
Deshalb besteht in Basic 7.0 die Möglichkeit, mit dem Befehl BEGIN den Anfang eines THEN- oder ELSE-Blockes zu definieren. An dessen Ende muß dann BEND stehen. Alles, was zwischen diesen Befehlen steht, wird dann als zusammenhängender THEN- oder ELSE-Block ausgeführt.
In Listing 2 (Abschnitt 2.3.2) werden BEGIN und BEND in den Zeilen 400-620 verwendet. Dort kann man gut sehen, daß Doppelpunkte am Anfang jeder Zeile zwischen BEGIN und BEND als Einrückungszeichen die Lesbarkeit beträchtlich erhöhen.

Wichtige Hinweise:

- Der BEGIN-Befehl muß nicht direkt hinter THEN oder ELSE stehen, also wäre auch »IF ... THEN PRINT:BEGIN« erlaubt.
- Steht BEGIN nach THEN, muß ein eventueller ELSE-Befehl hinter BEND stehen. (»BEND:ELSE («BEND:ELSE ...»). Man darf aber zwischen BEND und ELSE keine weiteren Befehle einsetzen, da dies zu Fehlfunktionen führt. Außerdem darf BEND nur am Zeilenende oder vor :ELSE» plaziert werden!
- Bei der Besprechung des ELSE-Kommandos wurde die Konstruktion

  ```
  IF...THEN...:ELSE IF...THEN...:ELSE
  ```

  vorgestellt. Wird diese in Verbindung mit BEGIN/BEND eingesetzt, so kann es Probleme geben, wenn zwar nach dem ersten THEN-Befehl ein BEGIN-/BEND-Block steht, aber die zweite IF...THEN...ELSE-Abfrage nicht durch BEGIN und BEND eingerahmt ist. Folgendes Programm ist also eine korrekte Umsetzung des Beispiels aus der ELSE-Erklärung:

  ```
  10 INPUT "ZAHL";Z:IF Z<0 THEN BEGIN
  20 :PRINT "DIE ZAHL IST NEGATIV"
  30 BEND:ELSE BEGIN
  40 :IF Z=0 THEN BEGIN:REM Schachtelung von BEGIN/BEND ist zugelassen
  50 ::PRINT "DIE ZAHL IST NULL"
  60 :BEND:ELSE BEGIN
  70 ::PRINT "DIE ZAHL IST POSITIV"
  80 :BEND
  90 BEND
  ```

  Falsch wäre hingegen folgendes Programm:

  ```
  10 INPUT "ZAHL";Z:IF Z<0 THEN BEGIN
  20 :PRINT "DIE ZAHL IST NEGATIV"
  30 BEND:ELSE IF Z=0 THEN BEGIN
  40 :PRINT "DIE ZAHL IST NULL."
  50 BEND:ELSE BEGIN
  60 :PRINT "DIE ZAHL IST POSITIV"
  70 BEND
  ```

- Wenn eine BEND-Anweisung fehlt, wird dies durch die Fehlermeldung

  ```
  ?BEND NOT FOUND ERROR
  ```

  signalisiert, die im Handbuch nicht erwähnt wird, aber durchaus vorhanden ist, wie die Direktmodus-Eingabe »IF 0 THEN BEGIN« beweist.

DO, LOOP, EXIT, WHILE und UNTIL – PASCAL-ähnliche Schleifen

Nun werden die Paradebefehle des C128 zum strukturierten Programmieren besprochen. Mit DO wird der Anfang einer Schleife, mit LOOP das Ende markiert:

```
DO:PRINT "*";:LOOP
```

gibt unzählig viele Sternchen aus. In dieser Form handelt es sich um eine Endlosschleife, da die DO-LOOP-Schleife nie verlassen wird, solange Sie nicht durch <RUN/STOP> abhelfen.
In Basic 2.0 ließe sich dies nur mit Hilfe eines GOTO-Befehls programmieren:

```
10 PRINT "*";:GOTO 10
```

Selbstverständlich sind auch anspruchsvollere Schleifen mit DO/LOOP möglich. Dafür benötigt man dann noch einen Befehl, der zum Verlassen der DO-LOOP-Schleife dient: »EXIT« setzt ein Programm mit dem nächsten auf »LOOP« folgenden Befehl fort. Wir wollen die Anweisung

```
FOR I=1 TO 10:PRINT I:NEXT I:REM Zahlen von 1 bis 10 ausgeben
```

in ein Basic-7.0-Programm mit DO, LOOP und EXIT übersetzen:

```
10 I=1
20 DO
30 :IF I>10 THEN EXIT
40 :PRINT I
50 :I=I+1
60 LOOP
```

(Die Einrückung des Schleifenrumpfes, also des Programmteils, der zwischen »DO« und »LOOP« steht, mit Doppelpunkten ist kein syntaktisches Erfordernis, erhöht die Lesbarkeit aber beträchtlich. Deshalb habe ich beim Erstellen der Listings zu diesem Buch ausgiebigen Gebrauch davon gemacht, damit Sie meine Beispielprogramme besser durchleuchten können.)
Der EXIT-Befehl ist aber nicht die einzige Möglichkeit, eine Schleife zu beenden. Dazu dient auch das Schlüsselwort WHILE, das entweder hinter DO, also am Schleifenanfang, oder hinter LOOP, also am Schleifenende, stehen muß. Nach WHILE steht eine Bedingung wie beim IF-Kommando; ist diese erfüllt, wird die Schleife fortgesetzt, ansonsten wird sie abgebrochen. »While« heißt nämlich »solange«.
Unser Beispielprogramm sieht also mit WHILE folgendermaßen aus:

```
10 I=1
20 DO WHILE I<=10
30 :PRINT I
40 :I=I+1
50 LOOP
```

Achtung: Zwischen DO bzw. LOOP und WHILE steht kein Doppelpunkt, sondern höchstens ein Leerzeichen!
Da die WHILE-Abfrage auch hinter LOOP stehen kann, ist ebenso folgendes möglich:

```
10 I=1
20 DO
30 :PRINT I
40 :I=I+1
50 LOOP WHILE I<=10
```

Es ist zwar in diesem Fall bedeutungslos, wo die WHILE-Anweisung plaziert ist; wenn wir aber Zeile 10 in »10 I=100« ändern, zeigt sich der Unterschied:
»WHILE« am Schleifenbeginn (also bei »DO«) beendet die Schleife sofort, da der Wert 100 als zu groß erkannt wird; »LOOP WHILE...« läßt dagegen die Schleife ein einziges Mal durchlaufen, da erst in Zeile 50 der Schleifenabbruch erfolgt.
Welche Lösung im Einzelfall vorzuziehen ist, muß von Fall zu Fall entschieden werden.
Anstatt Schleifen so lange durchlaufen zu lassen, als eine Bedingung wahr ist (»while«), kann man auch dem Schleifenrumpf wiederholen lassen, »bis« (engl. »until«) eine Bedingung eintritt:

```
10 I=1
20 DO UNTIL I>10
30 :PRINT I
40 :I=I+1
50 LOOP
```

Diese Lösung unterscheidet sich vom EXIT-Beispiel nur dadurch, daß anstelle des »IF I>10 THEN EXIT« die Abbruchbedingung »I>10« hinter UNTIL steht: »DO UNTIL I>10«.

Auch der UNTIL-Befehl darf auf LOOP folgen:

```
10 I=1
20 DO
30 :PRINT I
40 :I=I+1
50 LOOP UNTIL I>10
```

Für die Entscheidung, wo UNTIL besser aufgehoben ist, gilt das für den WHILE-Befehl Gesagte.

Der Vollständigkeit halber sei noch folgendes erwähnt:

- Die Bedingung zur Schleifenkontrolle (WHILE/UNTIL-Bedingung) kann auch logisch verknüpft sein (AND, OR, NOT).
- In jeder DO-LOOP-Schleife - ob UNTIL, WHILE oder keines von beidem vorkommt - können auch zusätzlich EXIT-Anweisungen in beliebiger Anzahl auftauchen. Zu häufiger Gebrauch dieses Befehls läßt ein Programm aber unübersichtlich werden.
- Während »FOR I%=1 TO 10« nicht erlaubt ist, da Integer-Variablen als FOR-NEXT-Schleifenzähler nicht zugelassen sind, können diese in DO-LOOP-Schleifen ohne weiteres verwendet werden (ersetzen Sie in unseren Beispielen »I« durch »I%«!). Sogar Stringvariablen (!) und Elemente aus Arrays, wie z.B. A(5), werden von DO/LOOP verarbeitet.
- DO-LOOP-Schleifen können nach denselben Regeln geschachtelt werden wie FOR-NEXT-Schleifen.
- UNTIL und WHILE können sowohl am Anfang als auch am Ende einer Schleife stehen, sie können sogar zweimal (einmal am Anfang, einmal am Ende) erscheinen und gemischt werden:

```
10 I=0
20 DO UNTIL I>10:REM Einmal nehmen wir "UNTIL"
30 :PRINT I
40 :I=I+1
50 LOOP WHILE I<=10:REM und einmal "WHILE"
```

Fehlermeldungen bei DO-LOOP-Schleifen

Bei der Arbeit mit DO/LOOP können zwei spezielle Fehler auftreten:

- LOOP NOT FOUND (DO ohne LOOP)
  Zu einer DO-Anweisung konnte das zugehörige Schleifenende nicht ermittelt werden, da der LOOP-Befehl fehlt.
- LOOP WITHOUT DO (LOOP ohne DO)
  Eine LOOP-Anweisung wurde gefunden, ohne daß eine DO-Anweisung vorausging.

Die Grenzen der Strukturbefehle des Basic 7.0

Sicherlich sind die vorgestellten Befehle schon ein Schritt weg vom Spaghetti-Code und ein Schritt hin zur strukturierten Programmierung. Dennoch fehlt dem Basic 7.0 zu einer »strukturierten Sprache« im eigentlichen Sinn die Eigenschaft, Prozeduren zu definieren, wie es etwa in PASCAL, MODULA, C und anderen Sprachen dieser Art möglich ist. Prozeduren sind einmal definierte Programmblöcke, die später unter einem (in der Regel sinnvoll gewählten) Namen aufgerufen werden können.
Dies ist in Basic 7.0 deswegen schwierig, weil es sich stark an Zeilennummern (statt an Namen) orientiert. Für die Strukturfreunde unter den Lesern sei deshalb ein Programm vorgestellt, das die Angabe von Sprungzielen durch Label ermöglicht, wie dies in anderen Programmiersprachen möglich ist.

Das Programm »LABEL 128«

Diese Maschinenroutine laden und aktivieren Sie über folgenden Befehl:

```
RUN "LABEL 128"
```

Zur Information: das Programm belegt den Bereich von 4864 ($1300) bis 5047 ($13B7) in Bank 0; der Basic-Anfang wird von 7168 ($1C00) nach 8192 ($2000) verlegt.
Dann stehen einige Alternativen zur herkömmlichen Methode, Sprungziele für GOTO und GOSUB mit feststehenden Zeilennummern anzugeben, zur Verfügung:

- GOTO/GOSUB auf eine errechnete Zeile (GOTO x/GOSUB x)
  Sprungziele für GOTO/GOSUB können jetzt so angegeben werden wie die numerischen Parameter anderer Befehle, d.h. auch Rechenausdrücke und Variablen (sogar Array-Variablen) sind gestattet:

```
X=1500:GOTO X
GOTO 15*100
```

```
C=15:GOSUB C*100
GOTO A(5)
GOSUB A(I)
```

Auf diese Weise lassen sich »ON...GOTO« und »ON...GOSUB« vermeiden.

- GOTO/GOSUB auf Label (Sprungmarken)
  Label müssen in einer REM-Zeile stehen. Folgendes ist möglich:

```
10 GOTO "LABEL"
...
...
...
1000 REMLABEL ← wichtig: kein Leerzeichen zwischen REM und Label!
```

Wenn Sie zwischen REM und Label ein Leerzeichen einfügen, muß dies auch nach dem ersten Anführungszeichen des Labels stehen:

```
10 GOTO "LABEL"
...
...
...
1000 REM LABEL
```

Gleiches gilt für GOSUB.

- GOTO/GOSUB auf Label in Stringvariablen
  Wenn in einer Stringvariablen ein Label enthalten ist, kann man auch folgendes verwenden:

```
10 L$="LABEL"
20 GOTO L$
...
...
...
1000 REMLABEL
```

Die Stringvariable kann auch Element eines Arrays sein:

```
10 CLR:DIM L$(100):L$(20)="LABEL"
20 GOTO L$(20)
...
...
...
1000 REMLABEL
```

Als Beispiele dienen noch fünf Demoprogramme.

Beschreibung: Demoprogramm 1 für »LABEL 128«
Filename: »label 128.demo 1«

```
10 REM
15 PRINTCHR$(147)
20 FOR I = 1 TO 10
30 GOSUB"ZAHL"
40 GOSUB"QUADRAT"
50 GOSUB"WURZEL"
60 NEXT
70 END
100 REMZAHL
110 PRINTI,
120 RETURN
300 REMWURZEL
310 PRINTSQR(I)
320 RETURN
510 REMQUADRAT
520 PRINTI*I,
530 RETURN
```

Beschreibung: Demoprogramm 2 für »LABEL 128«
Filename: »label 128.demo 2«

```
10 REM
20 PRINTCHR$(147)
30 Z$="ZAHL":Q$="QUADRAT":W$="WURZEL"
40 FOR I = 1 TO 10
50 GOSUBZ$:GOSUBQ$
70 GOSUBW$
80 NEXT
90 END
100 REMZAHL
110 PRINTI,
120 RETURN
300 REMWURZEL
310 PRINTSQR(I)
320 RETURN
510 REMQUADRAT
520 PRINTI*I,
530 RETURN
```

Beschreibung: Demoprogramm 3 für »LABEL 128«
Filename: »label 128.demo 3«

```
10 REM
15 TRAP1000
20 PRINTCHR$(147)
30 INPUT"ZAHL 1,2,3,4,5";Z
40 GOTOZ*100
50 END
100 PRINT"ZEILE 100":GOTO30
200 PRINT"ZEILE 200":GOTO30
300 PRINT"ZEILE 300":GOTO30
400 PRINT"ZEILE 400":GOTO30
500 PRINT"ZEILE 500":GOTO30
999 END
1000 RESUME 30
```

Beschreibung: Demoprogramm 4 für »LABEL 128«
Filename: »label 128.demo 4«

```
10 REM
15 TRAP1000
16 A$(1)="Z100":A$(2)="Z200":A$(3)="Z300"
17 A$(4)="Z400":A$(5)="Z500"
20 PRINTCHR$(147)
30 INPUT"ZAHL 1,2,3,4,5";Z
40 GOTOA$(Z)
50 END
100 REMZ100
110 PRINT"ZEILE 100":GOTO30
200 REMZ200
210 PRINT"ZEILE 200":GOTO30
300 REMZ300
310 PRINT"ZEILE 300":GOTO30
400 REMZ400
410 PRINT"ZEILE 400":GOTO30
500 REMZ500
510 PRINT"ZEILE 500":GOTO30
999 END
1000 RESUME 30
```

Beschreibung: Demoprogramm 5 für »LABEL 128«
Filename: »label 128.demo 5«

```
10 REM
20 TRAP 280
30 PRINT CHR$(19);CHR$(19):SCNCLR
40 TB=2-20*(RWINDOW(2)=80)
50 PRINT TAB(TB)"KOMMANDO-INTERPRETER MIT "CHR$(34)"LABEL 128"CHR$(34)
60 PRINT TAB(TB)"====================================="
70 PRINT:PRINT
80 PRINT TAB(TB-2)"$ WANDELT HEXADEZIMALZAHL IN DEZIMALZAHL"
90 PRINT TAB(TB-2)"# WANDELT DEZIMALZAHL IN HEXADEZIMALZAHL"
100 PRINT:PRINT
110 PRINT TAB(TB-2)"BEISPIEL: $FFD2 ERGIBT DEZIMALWERT"
120 PRINT TAB(TB+8)"#4096 ERGIBT HEXADEZIMALWERT"
130 PRINT:PRINT CHR$(27);"T";
140 PRINT "KOMMANDO:";
150 OPEN 1,0:INPUT#1,A$:CLOSE 1
160 K$=LEFT$(A$,1):REM KOMMANDO AUSSONDERN
170 P$=RIGHT$(A$,LEN(A$)-1):REM PARAMETER AUSSONDERN
180 GOSUB K$
190 GOTO 140
200 REM UNTERPROGRAMME
210 REM#
220 PRINT CHR$(29);" = $ ";HEX$(VAL(P$))
230 RETURN
240 REM$
250 PRINT CHR$(29);" = #";DEC(P$)
260 RETURN
270 :
280 REM FEHLERBEHANDLUNG
290 PRINT:RESUME 140
```

Da die Programme selbsterklärend sind, kann auf eine ausführliche Beschreibung verzichtet werden. Es sei nur erwähnt, daß sich Demo 2 von Demo 1 nur durch die Verwendung von Stringvariablen unterscheidet. Demo 4 hat gegenüber Demo 3 die Besonderheit, daß Array-Variablen zur Angabe der Label verwendet werden. Demo 5 setzt die Kenntnis des Hexadezimalsystems voraus.

Zum Abschluß die Einschränkungen bei der Arbeit mit »LABEL 128«:

- Das erweiterte GOTO/GOSUB ist nur im Programm verwendbar, aber nicht im Direktmodus!
- Folgt ein Sprung auf einen Label direkt auf THEN/ELSE, ist GOTO erforderlich:

  ```
  IF...THEN:GOTO "LABEL"
  IF...THEN...:ELSE:GOTO "LABEL"
  ```

  Sprünge auf feststehende Zeilennummern dürfen aber auch ohne GOTO/GOSUB angegeben werden (»IF ... THEN 50«); bereits bestehende Programme müssen also nicht geändert werden.
- Label dürfen nur bei GOTO/GOSUB verwendet werden, nicht bei anderen Befehlen (auch nicht bei anderen Befehlen, die die Angabe einer Zeilennummer erfordern, wie etwa TRAP).
- Bei der Eingabe sind Zeilennummern selbstverständlich weiterhin erforderlich, da »Label 128« keinen eigenen Editor hat.

3.4.3 Befehlsgruppe »Grafik«

Die Grafikbefehle des Basic 7.0 gehören zu dessen stärksten Seiten. In diesem Abschnitt wollen wir sie zuerst kennenlernen und dann einige verständnisfördernde Anwendungen dieser Befehle besprechen. Darunter finden sich recht schöne Grafikprogramme, die sicher auch den absoluten Grafikfreaks gefallen werden.

3.4.3.1 Die Grafikbefehle

Hier werden nicht alle Befehle und Grundlagen erklärt, sondern nur ein Überblick gegeben. Wir werden uns dabei aber auf das C128-Handbuch stützen, soweit es geht. Dies ist sicher auch im Sinne derer, die die Grafikbefehle bereits kennen und möglichst bald zu den Anwendungen kommen wollen.
Lesen Sie bitte, falls Sie es noch nicht getan haben, im Handbuch die Seiten 4-135 bis 4-143. Dann kennen Sie die folgenden Befehle:

COLOR	Farbquellen mit Farbcodes (1-16) belegen
GRAPHIC	Grafikmodus anwählen
LOCATE	Pixel-Cursor positionieren
CIRCLE	Kreis, Ellipse oder Vieleck (Polygon) zeichnen

BOX	Rechteck zeichnen und auf Wunsch ausfüllen
DRAW	Punkt(e) oder Linie(n) zeichnen
PAINT	Fläche ausfüllen
CHAR	Text in hochauflösende Grafik schreiben
SCALE	Grafikmaßstab verändern

Zu diesen Befehlen empfiehlt es sich, auch die Befehlsbeschreibung in Kapitel 4.6 des Handbuches zu lesen. Die restlichen Grafikbefehle werden wir uns jetzt gemeinsam erarbeiten. Vorher sollen aber noch Möglichkeiten des DRAW-Befehls, die das Handbuch verschweigt, erwähnt werden; diese gelten teilweise auch für andere Grafikbefehle wie BOX und PAINT.

Koordinatenangaben bei DRAW und anderen Grafikbefehlen

Außer der üblichen Methode, X- und Y-Koordinate eines Punktes mitzuteilen, gibt es noch zwei weitere Möglichkeiten:

a) Relativkoordinaten

Wie Sie aus dem Handbuch wissen, merkt sich der C128 den letzten gezeichneten oder gelöschten Punkt als Pixel-Cursor. Dieser kann bekanntlich auch mit LOCATE eingestellt werden. Nun ist es möglich, ausgehend von der aktuellen Pixel-Cursorposition, die Koordinaten eines Punktes anzugeben. Sehen wir uns folgendes Beispielprogramm an:

```
100 GRAPHIC 1,1:REM Grafik ein & löschen
110 LOCATE 160,100:REM Pixel-Cursor in Bildschirm-Mitte
120 DRAW 1, TO +50,-50
```

Dieses Programm zeichnet eine Linie zu dem Punkt, dessen X-Koordinate um 50 größer ist als die Position des Pixel-Cursors, während die Y-Koordinate um 50 kleiner ist (dies erkennt der Computer daß die Zeichen »+« und »–« vor der Zahl stehen). Folglich wird, da der Pixel-Cursor bei 160,100 liegt (Zeile 110), eine Linie zum Punkt 210,50 gezogen.
Beachten Sie den Unterschied: »DRAW 1,50,50« zeichnet den Punkt 50,50!
MOVSPR, ein Befehl zur Spriteprogrammierung, die uns auch noch beschäftigen wird, arbeitet ebenfalls mit Relativkoordinaten.

b) Polarkoordinaten

Auch diese Form der Koordinatenangabe geht vom Pixel-Cursor aus. Es wird aber nicht die Verschiebung in X- und Y-Richtung angegeben, sondern Winkel und Abstand vom Pixel-Cursor. Diese Koordinatenangabe wird daran erkannt, daß anstelle eines Kommas ein Semikolon (»;«) steht, auf das der Winkel und der Abstand folgen.
Folgendes Programm zeichnet einen Punkt, der vom Punkt 160,100 im 30-Grad-Winkel 100 Pixel entfernt ist:

```
100 GRAPHIC 1,1:REM Grafik ein & löschen
110 LOCATE 160,100:REM Pixel-Cursor positionieren
120 DRAW 1;30,100
```

Die Grad-Angabe des Richtungswinkels entspricht der Kompaßrose.

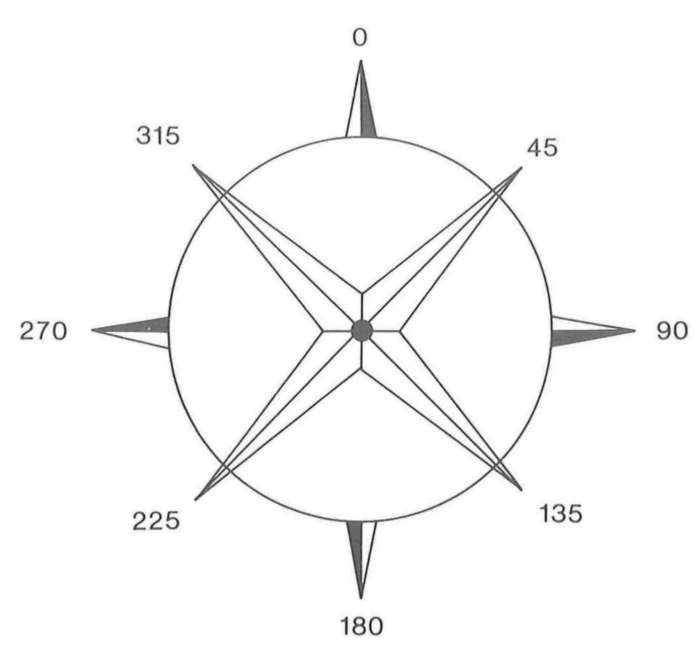


Bild 3.1: *Die Kompaßrose*

Syntax der Grafikbefehle

Manche Parameter bei Grafikbefehlen können zwar nicht einfach entfallen, sind aber bereits sinnvoll vorbelegt. So ist z.B. als Farbquelle der Wert 1 (Punkt in Zeichenfarbe setzen) eingestellt. Diese Voreinstellungen, die in der Befehlsbeschreibung in Kapitel 4.6 des Handbuches aufgeführt sind, erreicht man, indem man einen Parameter nicht angibt, aber ein Komma setzt, damit das Basic erkennt, daß ein Parameter ausgelassen wurde:
Statt

```
DRAW 1,100,100
```

kann man auch

```
DRAW,100,100
```

schreiben.

Weitere Grafikbefehle

Außer den in Kapitel 4.7 des C128-Handbuches angesprochenen Befehlen existieren noch weitere, die wir hier erklären wollen. Diese Anweisungen stehen auch im Kapitel 4.6 des Handbuches.

RGR - Das Gegenstück zu GRAPHIC

Den mit »GRAPHIC n« eingestellen Grafikmodus kann man auf einfache Weise mit

```
PRINT RGR(0)
```

ermitteln, auch wenn der GRAPHIC-Befehl nicht verwendet wurde.
Dabei kann statt 0 ein beliebiger anderer Wert stehen, eine Zahl muß aber angegeben werden; dies kennen Sie schon von der FRE-Funktion in Basic 2.0.
Das Ergebnis von RGR(0) ist immer eine Zahl im Bereich 0-5, die Bedeutung ist dieselbe wie hinter »GRAPHIC«.

RCLR - Das Gegenstück zu COLOR

RCLR liefert den Wert einer Farbquelle. Die Farbquelle wird wie beim COLOR-Befehl angegeben, ist also ein Wert von 0 bis 6.

```
PRINT RCLR(1)
```

gibt die Zeichenfarbe in der hochauflösenden Grafik aus. Der von RCLR erzeugte Wert liegt im Bereich 1-16, da es sich um den Farbcode handelt, wie er bei COLOR angegeben wird.

RDOT - Pixel testen

Wenn man feststellen will, ob ein Punkt in der hochauflösenden Grafik gesetzt ist, positioniert man zuerst den Pixel-Cursor mit LOCATE an diesem Punkt (LOCATE x,y) und ruft dann die RDOT-Funktion auf:

```
PRINT RDOT(2)
```

liefert die Farbquelle: 0 = Punkt gelöscht (Hintergrundfarbe)
1 = Punkt gesetzt (Vordergrundfarbe)
2 = Punkt gesetzt (Multicolorfarbe 1)
3 = Punkt gesetzt (Multicolorfarbe 2)

Man kann aber durch Angabe eines anderen Parameters als 2 mit RDOT die Position des Pixel-Cursors ermitteln:

`PRINT RDOT(0)` liefert die aktuelle X-Koordinate der Position,
`PRINT RDOT(1)` liefert die aktuelle Y-Koordinate der Position.

WIDTH - Punktbreite bestimmen

Die Punktbreite (einfach oder doppelt) bei den Zeichenbefehlen (DRAW, BOX, CIRCLE usw.) kann mit WIDTH (englisches Wort für »Breite«) eingestellt werden:

WIDTH 1 einfache Strichstärke (1 Pixel)
WIDTH 2 doppelte Strichstärke (2 Pixel)

Alle nach WIDTH folgenden Zeichenbefehle richten sich dann nach dieser Einstellung. Deshalb gleicht »WIDTH 2« im Hochauflösungsmodus die Punktbreite der Multicolorgrafik an, was in bestimmten Fällen erwünscht sein kann.

SCNCLR – Grafik- und Textbildschirme löschen

Wenn man einen Grafik- oder Textbildschirm löschen will, ohne

```
GRAPHIC Modus,1
```

einzusetzen, ist der SCNCLR-Befehl dafür geeignet:

```
SCNCLR (Modus)
```

löscht den Bildschirm, der durch »Modus« bestimmt wird. Dieser Parameter entspricht der ersten Angabe nach dem GRAPHIC-Befehl.
Wird nur

```
SCNCLR
```

geschrieben, löscht der C128 den aktuellen Bildschirm, der mit der RGR-Funktion ermittelt werden kann.
SCNCLR eignet sich also auch im Textmodus; dort ist es ein Ersatz für den Befehl »PRINT CHR$(147)«, der in Basic 2.0 zum Löschen des Textbildschirms nötig ist.
Beispiel: SCNCLR(0) löscht den 40-Zeichen-Textbildschirm.
Hinweis: Im Gegensatz zu GRAPHIC wird bei SCNCLR der betreffende Modus nicht angewählt.

Die Spritebefehle in Basic 7.0

Auch die Sprites, die Sie vom C64 kennen und die deshalb nicht weiter erklärt werden müssen, werden von Basic-Befehlen unterstützt. Dies hat zur Folge, daß die vielen POKE-Befehle, die auf dem C64 nötig waren, durch überschaubare Basic-Kommandos ersetzt werden. Allerdings sind im 80-Zeichen-Modus keine Sprites möglich(!).
Gegenüber dem C64 hat sich geändert, daß die Sprites jetzt im Adreßbereich 3584-4095 ($0E00-$0FFF) liegen, der Raum für die 8 Sprites bietet, die von Basic 7.0 verwaltet werden. Wenn man die Sprites in diesem Bereich auf Diskette speichern willen, verwendet man folgendes Kommando:

```
BSAVE "SPRITES 1-8",ON B0,P3584 TO P4095
```

Geladen werden sie mit:

```
BLOAD"SPRITES 1-8",ON B0
```

SPRDEF – Der eingebaute Sprite-Editor

Im C64-Handbuch wird zwar ausführlich beschrieben, wie man ein Sprite-Muster in Zahlenwerte umrechnet; das Erstellen auf dem Papier ist aber sehr unkomfortabel. Ein einigermaßen

leistungsfähiger Sprite-Editor nimmt uns auf dem C128 diese Arbeit ab. Er wird über den Basic-Befehl

```
SPRDEF
```

aufgerufen, welcher sogar in Programmen stehen darf.
Sie sehen am 40-Zeichen-Bildschirm ein Definitionsfeld (24*21 Punkte), rechts ist das Sprite so dargestellt, wie es später aussehen wird.
Unmittelbar nach dem Start erscheint die Frage »SPRITE NUMBER?«. Diese wird durch Eingabe der Spritenummer (1-8) beantwortet. Alle späteren SPRITE-Befehle beziehen sich auf diese Nummer, weshalb es ratsam ist, mit 1 zu beginnen.
Falls das durch die Nummer definierte Sprite noch nicht editiert war, erscheint jetzt sogenannter »Sprite-Müll«, der mit <SHIFT>+<CLR/HOME> gelöscht wird; dann kann das Editieren beginnen.
Folgende Befehle stellt der Sprite-Editor zur Verfügung (Tabelle 3.2):

Tabelle 3.2: *Befehle des Sprite-Editors SPRDEF*

CURSORTASTEN:	Der Editor-Cursor wird mit den herkömmlichen Cursortasten bewegt.
<RETURN>:	Cursor springt an den Anfang der nächsten Zeile
<CLR>:	Das Definitionsfeld wird gelöscht
<HOME>:	Sprite-Editor-Cursor kommt in die linke obere Ecke des Definitionsfeldes
<M>:	Normalerweise sind alle 8 Sprites hochauflösend (24*21 Punkte); Multicolorsprites haben, ähnlich der Multicolorgrafik, nur die halbe Auflösung (12*21 Punkte), aber dafür 4 Farben (statt 2). Durch <M> wird zwischen Hochauflösungs- und Multicolor-Modus umgeschaltet.
<X>:	Schaltet zwischen einfacher und doppelter Ausdehnung in X-Richtung um.
<Y>:	Wie <X>, aber für Y-Ausdehnung.
FARBTASTEN:	Wie im Textmodus, wird die Vordergrundfarbe mit <CONTROL>+<Zifferntaste> bzw. <CBM>+<Zifferntaste> eingestellt.
<1>-<4>:	Im Hochauflösungsmodus (Normalmodus): <1> setzt Hintergrundfarbe im Definitionsfeld <2> setzt Vordergrundfarbe Im Multicolormodus (siehe <M>): <1> setzt Hintergrundfarbe <2> setzt Multicolorfarbe 1 <3> setzt Vordergrundfarbe <4> setzt Multicolorfarbe 2
<A>:	Mit <A> kann die Wiederholungsfunktion der Tasten <1>-<4> abgestellt werden, nochmaliges <A> stellt wieder den Ausgangszustand her.

<C> oder <F1>: Auf die Frage »COPY FROM?« muß die Nummer des Sprites eingegeben werden, welches in das Definitionsfeld kopiert werden soll. Mit <RETURN> kommt man von dieser Funktion zurück, ohne daß ein Kopiervorgang ausgelöst wird.

<CONTROL>+<C>: Diese Kombination ermöglicht ein Umschalten zwischen den Sprites 1-8, indem man die neue Spritenummer eingibt.

<STOP>: Beantwortet man die Frage »SPRITE NUMBER?« mit einer Zifferntaste (1-8), wird dieses Sprite aus dem Spritebereich (3584-4095) geholt und kann editiert werden. Mit <RETURN> wird der SPRITE-Editor verlassen.

<SHIFT-RETURN>: Das Definitionsfeld wird in den Sprite-Speicher übertragen. Dann geht es weiter wie bei <STOP>.

Die Befehle <C> bzw. <F1> und <CONTROL>+<C> werden im Handbuch nicht beschrieben.
Im Sprite-Editor gleichen die Funktionstasten folgenden Eingaben:
<F1>=<C>, <F2>=<A>, <F5>=<A>, <F6>=<RETURN>, <F7>=<RETURN>, <F8>= <M>

SPRCOLOR - Multicolorfarben für Sprites setzen

Bei Verwendung von Multicolorsprites setzt man die beiden Multicolorfarben über folgenden Befehl:

```
SPRCOLOR Multicolorfarbe 1 , Multicolorfarbe 2
```

Die Farben werden wie bei COLOR angegeben. Beispiel:

```
SPRCOLOR 7,16
```

setzt die Farbe Blau als Multicolorfarbe 1,
die Farbe Hellgrau als Multicolorfarbe 2.

SPRITE - Sprite einschalten und Eigenschaften festlegen

Voraussetzung für diesen Befehl ist, daß das Spritemuster schon im Spritespeicher steht. Dies ist der Fall, wenn es im Sprite-Editor mit <SHIFT>+<RETURN> dort abgelegt wurde, oder wenn der Spritebereich von Diskette geladen wurde. Sorgen Sie jetzt bitte dafür, daß ein »brauchbares« Sprite 1 im Speicher steht.
Dann können Sie dieses mit dem Befehl SPRITE definieren:

```
SPRITE N,EA,V,P,XA,YA,M
```

Dabei ist:

N Spritenummer (1-8)
EA Sprite ausschalten (EA=0) oder einschalten (EA=1)

V	Vordergrundfarbe (1-16)
P	Priorität: Sprite über Text (P=0) oder Text über Sprite (P=1) Die Priorität von Sprite zu Sprite ergibt sich aus der Spritenummer: Sprite 1 über Sprite 2, Sprite 2 über Sprite 3 usw.
XA	X-Ausdehnung: normal (XA=0) oder doppelt (XA=1)
YA	Y-Ausdehnung: normal (YA=0) oder doppelt (YA=1)
M	Modus: normales Hochauflösungssprite (M=0) oder Multicolorsprite (M=1)

Nun können Sie für jedes Sprite die entsprechenden Werte einsetzen.

MOVSPR - Sprites positionieren und bewegen

Hat der SPRITE-Befehl das Sprite noch nicht sichtbar gemacht? Dann müssen Sie erst dem C128 mitteilen, wo das Sprite am Bildschirm stehen soll. Dies geschieht über

```
MOVSPR Spritenummer,X-Koordinate,Y-Koordinate
```

Beispiel: `MOVSPR 1,100,100`

Aber Achtung: die Spritekoordinaten entsprechen nicht den Koordinaten in der hochauflösenden Grafik, sondern haben ein eigenes Schema, das Sie vielleicht schon vom C64 kennen:

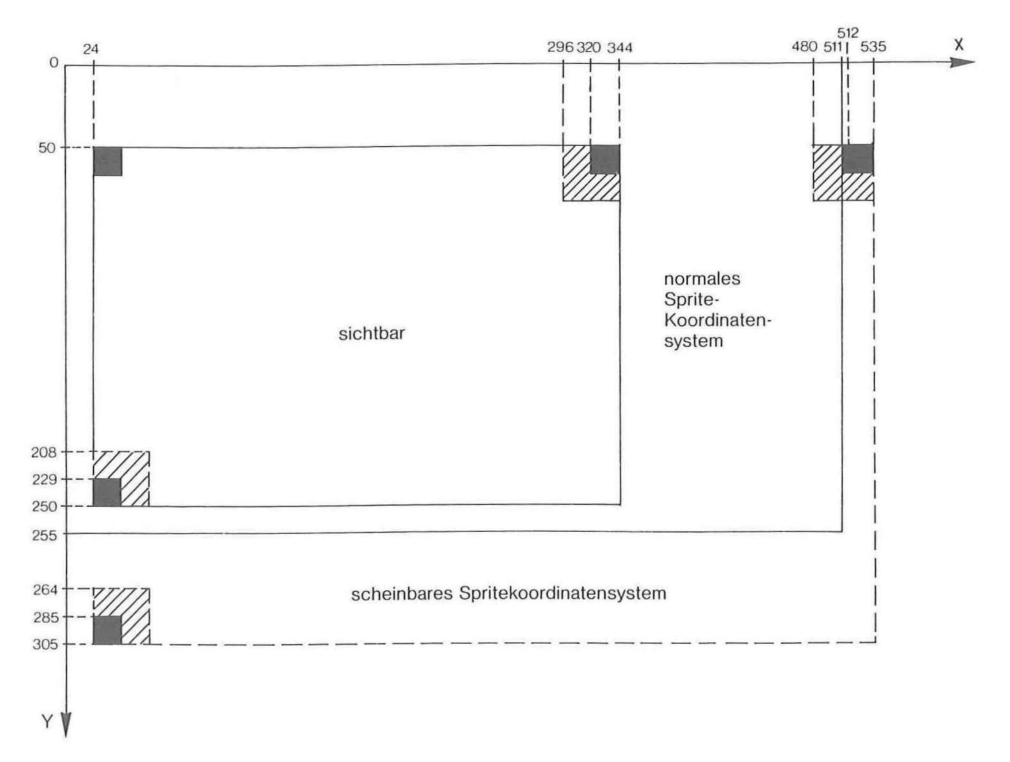


Bild 3.2: *Das Sprite-Koordinatensystem*

Der sichtbare Bildschirm reicht in X-Richtung von 24 bis 344, in Y-Richtung von 50 bis 250. Basic 7.0 verfügt nicht über die Möglichkeit, Sprites am Bildschirmrand (!) darzustellen, wie dies einige interessante Maschinenprogramme für den C64 können. Dies wäre aber auch eine zu große Anforderung; Profis, die unbedingt wissen wollen, wie das geht, sei das Programm »Der Riesen-Bildschirm« für den C64 (bzw. C64-Modus des C128) empfohlen, das in der Computerzeitschrift 64'er in der Ausgabe 1/86 auf Seite 80 veröffentlicht wurde.
Zurück zu MOVSPR. Außer der Positionierung ist auch noch die Bewegung von Sprites möglich. Hier gibt es zwei verschiedene Arten von Animation:
a) Bewegung von der aktuellen Position an Zielposition
b) Bewegung in bestimmter Richtung parallel zum laufenden Programm (!)
Beides leistet MOVSPR. Mit

```
MOVSPR Spritenummer,+/- X,+/- Y
```

wird das Sprite an die Zielposition bewegt, die in Form von Relativkoordinaten angegeben wird, wie wir es bei den Befehlen zur hochauflösenden Grafik kennengelernt haben.

Beispiel: `MOVSPR 1,-10,+20`

Ein beeindruckender Effekt wird durch

```
MOVSPR Spritenummer, Winkel#Geschwindigkeit
```

Die Bewegungsrichtung wird wieder in Grad angegeben, wofür die Kompaßrose gilt, die weiter vorne in diesem Abschnitt abgebildet ist (bei der Erklärung von Polarkoordinaten bei Grafikbefehlen).
Die Geschwindigkeit wird durch einen Wert von 0 (niedrigste Geschwindigkeit) bis 15 (höchste Geschwindigkeit) festgelegt. Zwischen Winkel und Geschwindigkeit steht ein Doppelkreuz.

Beispiel: `MOVSPR 1,180#10`
bewegt Sprite 1 abwärts mit der Geschwindigkeit 10;
das laufende Programm wird sofort fortgesetzt.

Damit haben wir die grundlegenden Spritebefehle besprochen. Folgende Befehle gibt es außerdem noch, deren Bedeutung Sie bitte der Basic-7.0-Befehlsübersicht im C128-Handbuch entnehmen:

```
SPRSAV,COLLISION,BUMP,RSPRITE,RSPCOLOR,RSPPOS
```

Diese Befehle sind zwar auch recht interessant, werden aber im Handbuch in ausreichender Ausführlichkeit beschrieben, was eine Beschreibung im Rahmen dieses Buches überflüssig macht.

Shapes - Spezialität des Basic 7.0

Außer Sprites kann das Basic 7.0 bestimmte Bildschirmbereiche als sogenannte »Shapes« (Formen) definieren. Diese können allerdings nur in der hochauflösenden Grafik verwendet werden und sind wesentlich langsamer als Sprites.

Die dazu benötigten Befehle heißen GSHAPE und SSHAPE und werden im Handbuch auf den Seiten 4-144 bis 4-152 zusammen mit einigen Spriteanweisungen erklärt. Wir wollen aber nun zu den Anwendungen der Grafikbefehle des Basic 7.0 kommen und uns nicht länger mit Befehlserklärungen aufhalten.

3.4.3.2 Grafik-Beispiele und Anwendungen in Basic 7.0

Anwendung 1: Apfelberge (Fractalberge)

Zu den erstaunlichsten Grafikeffekten gehört die sogenannte »Mandelbrotmenge«; dies ist ein mathematisches Verfahren, das interessante Grafiken erzeugt.
Der Mandelbrot-Algorithmus beruht im Prinzip darauf, daß in eine Formel immer das Ergebnis der letzten Berechnung eingesetzt wird. Mehr soll dazu aber nicht gesagt werden, denn die Grundlagen werden im Buch »Grafikprogrammierung C128«, das schon erwähnt wurde, vermittelt. Ein ähnliches Programm für den C64 wurde im 64'er-Magazin, Ausgabe 11/85, auf Seite 80 unter der Überschrift »Bilder aus einer anderen Dimension« vorgestellt. In beiden genannten Quellen wird ein Programm beschrieben, um zweidimensionale »Apfelmännchen« (so heißen diese Grafiken) zu erzeugen.
Neu ist aber die (noch eindrucksvollere) 3D-Darstellung von Fractals, die von einem Programm, das wir noch besprechen werden, ermöglicht wird. Damit Sie gleich einen Eindruck bekommen, wie so ein fertiger »Apfelberg« aussieht, legen Sie bitte die Programmdiskette ein und geben Sie

```
RUN "LOAD APFELBERG"
```

ein. Dann fragt Sie dieses Programm nach 4 Farbcodes (1-16); am besten beantworten Sie diese Fragen mit den Werten 1,2,3 und 7. Als Filenamen geben Sie dann

```
APFELBERG.PIC
```

an. Das Programm lädt diese Grafik von Diskette und zeigt sie sofort an; mit der Linkspfeiltaste kommt man in den Textmodus zurück.
Beeindruckt? Dann wollen wir uns dem Programm zuwenden, das solche fantastischen Grafiken generiert.
Beschreibung: Apfelberge darstellen
Filename: »apfelberge«

```
100 REM ***      FRACTALBERGE        ***
110 REM ***      (APFELBERGE)        ***
120 REM *** MIT PARAMETER-EINGABE ***
130 :
140 COLOR 0,12:COLOR4,14:COLOR5,14
150 SCNCLR:GRAPHIC 1:GRAPHIC 0,1:SLOW
160 :
170 XC=1:YC=0:T=20:S=60:XL=-.15:XR=.26:YO=.47:YU=.9:XM=105:YM=105:HG=7:F1=8:F2=1
1:F3=1
180 :
190 PRINTTAB(10)"HINTERGRUND.....";HG
200 PRINTTAB(10)"FARBE 1.........";F1
210 PRINTTAB(10)"FARBE 2.........";F2
220 PRINTTAB(10)"FARBE 3.........";F3
```

```
230 PRINTTAB(10)"X KOMPLEX.......";XC
240 PRINTTAB(10)"Y KOMPLEX.......";YC
250 PRINTTAB(10)"RECHENTIEFE.....";T
260 PRINTTAB(10)"MAXIMALE HOEHE..";S
270 PRINTTAB(10)"X LINKS.........";XL
280 PRINTTAB(10)"X RECHTS........";XR
290 PRINTTAB(10)"Y OBEN..........";YO
300 PRINTTAB(10)"Y UNTEN.........";YU
310 PRINTTAB(10)"AUSMASS X.......";XM
320 PRINTTAB(10)"AUSMASS Y.......";YM
330 PRINT:PRINT "FUER DIESE WERTE MUSS MAN NUR IMMER DIE"
340 PRINT "RETURN-TASTE DRUECKEN."
350 PRINT
360 :
370 INPUT"HINTERGRUND";HG
380 INPUT"FARBE 1     ";F1
390 INPUT"FARBE 2     ";F2
400 INPUT"FARBE 3     ";F3
410 INPUT "X KOMPLEX";XC
420 INPUT "Y KOMPLEX";YC
430 INPUT "MAX.TIEFE";T
440 INPUT "MAX.HOEHE";S
450 INPUT "X LINKS";XL
460 INPUT "X RECHTS";XR
470 INPUT "Y OBEN";YO
480 INPUT "Y UNTEN";YU
490 DO:INPUT"AUSMASS X";XM:LOOP WHILE XM=0
500 DO:INPUT"AUSMASS Y";YM:LOOP WHILE YM=0
510 :
520 PRINT"ALLES RICHTIG (J/N) ?"
530 DO:GETKEY A$:LOOP UNTIL A$="J" OR A$="N"
540 IF A$="N" THEN RUN:ELSE DX=(XR-XL)/XM:DY=(YU-YO)/YM
550 :
560 PRINT"FAST-MODUS (J/N) ?"
570 DO:GETKEY A$:LOOP UNTIL A$="J" OR A$="N"
580 IF A$="J" THEN FAST
590 :
600 GRAPHIC 5,1
610 PRINT CHR$(2);"PARAMETER";CHR$(27);"O":PRINT
620 PRINTTAB(10)"HINTERGRUND.....";HG
630 PRINTTAB(10)"FARBE 1.........";F1
640 PRINTTAB(10)"FARBE 2.........";F2
650 PRINTTAB(10)"FARBE 3.........";F3
660 PRINTTAB(10)"X KOMPLEX.......";XC
670 PRINTTAB(10)"Y KOMPLEX.......";YC
680 PRINTTAB(10)"RECHENTIEFE.....";T
690 PRINTTAB(10)"MAXIMALE HOEHE..";S
700 PRINTTAB(10)"X LINKS.........";XL
710 PRINTTAB(10)"X RECHTS........";XR
720 PRINTTAB(10)"Y OBEN..........";YO
730 PRINTTAB(10)"Y UNTEN.........";YU
740 PRINTTAB(10)"AUSMASS X.......";XM
750 PRINTTAB(10)"AUSMASS Y.......";YM
760 :
770 GRAPHIC 1,1:GRAPHIC 0
780 GRAPHIC 3,1:COLOR 1,F1:COLOR 4,HG:COLOR 2,F2:COLOR 3,F3:COLOR 0,HG
790 FORN=0TOYM:Y1=YO+N*DY:FORM=0TOXM:X=XL+M*DX:Y=Y1:K=0
800 DO:X2=X*X:Y2=Y*Y:Y=2*X*Y-YC:X=X2-Y2-XC:K=K+1:LOOP WHILE (K<T)AND(X2+Y2<S)
810 U=M+53-N/2:U1=U+1:V=N+80:V1=V-3*(K-1)
820 DRAW3,U,VTOU,V1:DRAW2,U1,VTOU1,V1:DRAW1,U,V1TOU1,V1
830 NEXT:NEXT
840 SLOW
```

```
850 :
860 DO:GET A$:PRINT CHR$(27);"G";CHR$(7);:LOOP UNTIL A$="←"
870 GRAPHIC 0
880 :
890 :
900 COLOR 0,12:COLOR4,14:COLOR5,14
910 SCNCLR
920 :
930 PRINT "WAEHLEN SIE AUS:"
940 PRINT:PRINT"1:  GRAFIK EINBLENDEN
950 PRINT:PRINT"2:  GRAFIK ABSPEICHERN
960 PRINT:PRINT"3:  NEUE GRAFIK ERSTELLEN LASSEN"
970 PRINT:PRINT"BITTE TASTE DRUECKEN (1,2 ODER 3) !"
980 PRINT:PRINT:PRINT"NACH EINBLENDEN DER GRAFIK MUSS WIEDER"
990 PRINT"DIE LINKSPFEIL-TASTE (←) GEDRUECKT WER-"
1000 PRINT"DEN, UM IN DIESES KLEINE AUSWAHLMENUE ZU"
1010 PRINT"GELANGEN."
1020 PRINT:PRINT:PRINT"ABGESPEICHERTE GRAFIKEN KOENNEN MIT DEM"
1030 PRINT"PROGRAMM 'LOAD APFELBERG' WIEDER EINGE-"
1040 PRINT"LADEN UND BETRACHTET WERDEN."
1050 :
1060 DO:GETKEY A$:LOOP WHILE A$<"1" OR A$>"3"
1070 :
1080 IF A$="1" THEN GRAPHIC3:COLOR0,HG:COLOR4,HG:COLOR1,F1:COLOR2,F2:COLOR3,F3:D
O:GETKEY A$:LOOP UNTIL A$="←":GOTO 870
1090 :
1100 IF A$="3" THEN SCNCLR:GOTO 190
1110 :
1120 DO:INPUT "NAME DER GRAFIK";N$:LOOP WHILE LEN(N$)<1 OR LEN(N$)>16
1130 BSAVE (N$),ON B0,P(DEC("1C00")) TO P(DEC("3FFF"))
1140 PRINT"DISKMELDUNG: ";DS$
1150 IF DS THEN GOTO 1120:ELSE 910
```

Besprechen wir zunächst die Anwendung, dann die Funktionsweise des Programms.
Zur reinen Anwendung sollten Sie das Programm »c/apfelberge« (befindet sich nur auf der Programmdiskette) verwenden; es handelt sich hierbei um ein Compilat, d.h. eine Umsetzung des Basic-Programms in ein Maschinenprogramm, was eine wesentlich kürzere Ausführungszeit zur Folge hat. Dennoch bewegt sich die Rechenzeit für einen Apfelberg im Stundenbereich, da extrem hoher Rechenaufwand erforderlich ist, um die vielen Punkte zu zeichnen.
Nach dem Start des Programms werden die voreingestellten Parameter angezeigt und neue Parametereingaben gefordert. Drücken Sie bei einer Parameterangabe <RETURN>, so wird die Voreinstellung übernommen. Folgende Werte werden verlangt:

Hintergrund:	Hintergrundfarbe (1-16)
Farbe 1:	Zeichenfarbe 1 (1-16)
Farbe 2:	Zeichenfarbe 2 (1-16)
Farbe 3:	Zeichenfarbe 3 (1-16)
X komplex:	Zahl, die das Aussehen des Bildes bestimmt. Schon kleine Abweichungen bringen neue Bilder hervor.
Y komplex:	Ähnlich wie »X komplex«,aber ein anderer Parameter.
Rechentiefe:	Je größer die Rechentiefe, desto verschachtelter wird die Grafik. Werte über 200 ist abzuraten.
Maximale Höhe:	Höhe der Balken in Pixels.
X links,	Auch diese Parameter beeinflussen die Grafik.

X rechts,	Experimentieren Sie mit Werten, die von der
Y oben,	Voreinstellung nur geringfügig abweichen.
Y unten.	Werte zwischen -1 und +1 eignen sich besonders.
Ausmaß X:	Bestimmt die Breite der Grafik (Angabe in Pixels).
Ausmaß Y:	Bestimmt die Höhe der Grafik (Angabe in Pixels).

Für einen ersten Testlauf sollten Sie immer <RETURN> drücken und höchstens die Bildschirmfarben neu eingeben.
Nach der Eingabe der Zahlenwerte und der Bestätigungsfrage (»Alles richtig?«) kann der FAST-Modus eingestellt werden; dieser verringert die Rechenzeit um mehr als 50%, allerdings kann man dann nicht sehen, wie die Grafik Linie für Linie gezeichnet wird. Ist die Grafik fertig, wird wieder auf »SLOW« geschaltet, damit Sie das Ergebnis betrachten können.
Während die Grafik gezeichnet wird, kann man am 80-Zeichen-Bildschirm die Parameter sehen.
Da die Ausführungszeit mehrere Stunden beträgt (von Bild zu Bild verschieden), wird ein schriller Signalton erzeugt, sobald die Grafik fertig ist. Dieser macht es überflüssig, vor dem Bildschirm auszuharren.
Der Signalton wird mit der Linkspfeiltaste abgestellt, die dann zum Ausblenden der Grafik zugunsten eines Auswahlmenüs führt. Dort blendet Punkt 1 wieder die Grafik ein (Linkspfeil führt ins Menü zurück), Punkt 2 dient zum Abspeichern der Grafik und Punkt 3 führt zum Programmanfang. Voreingestellt sind dann die letzten eingegebenen Parameter.
Über Punkt 2 abgespeicherte Grafiken können mit dem Programm »LOAD APFELBERG«, das wir auch noch besprechen wollen, eingelesen und betrachtet werden. Dies geht wesentlich schneller als ein erneutes Zeichnen.
Besprechen wir nun die interessantesten Stellen im Programm »Apfelberge«.
Die Befehle »GRAPHIC 1:GRAPHIC 0,1« dienen dem Umschalten auf den 40-Zeichen-Textbildschirm.
In Zeile 170 werden die Parameter vorbelegt und in 190-350 angezeigt. Die Neueingabe erfolgt in den Zeilen 370-500. Bemerkenswert ist, daß in 490 und 500 eine DO-LOOP-Schleife zum Abfangen der Fehleingabe 0 eingesetzt wird; im Gegensatz zu »IF Fehleingabe THEN GOTO« benötigen DO und LOOP keine Zeilennummer, was die Programmierung vereinfacht.
Aus den eingegebenen Werten wird, wenn die Eingaben bestätigt wurden, in Zeile 540 die Berechnung der Konstanten DX und DY, die von der Zeichenroutine benötigt werden, vorgenommen.
Zeile 600 schaltet auf den 80-Zeichen-Bildschirm um, auf dem die Zeilen 610-750 die Parameter einblenden, bis Zeile 770 wieder in den Textmodus mit 40 Zeichen/Zeile geht.
Die eigentliche Zeichenroutine besteht aus den Zeilen 780-830. Da sie sehr kompliziert und nur mit entsprechenden mathematischen Grundlagen erklärt werden kann, wollen wir auf eine Beschreibung verzichten. Interessierte können in den genannten Quellen (Grafikprogrammierung C128 oder 64'er 11/85) nachlesen, wie der Mandelbrot-Algorithmus programmiert wird.
Mit Zeile 900 beginnt das Menü; in Zeile 1130 steht ein BSAVE-Befehl, der die Multicolorgrafik abspeichert. Die über COLOR eingestellten Farben werden nicht berücksichtigt, da diese variabel sein sollen.

Das Einladen von Apfelberg-Grafiken ermöglicht das kurze Programm

"LOAD APFELBERG".

Beschreibung: Apfelberg-Grafiken einlesen und anzeigen
Filename: »load apfelberg«

```
100 REM *** LOAD APFELBERG ***
110 :
120 PRINT:PRINT "LOAD APFELBERG (APFELBERG LADEN)"
130 PRINT:PRINT"VOR DEM LADEN WIRD DIE GRAFIK BEREITS"
140 PRINT:PRINT"EINGESCHALTET. DURCH DRUECKEN DER LINKS-"
150 PRINT:PRINT"PFEILTASTE '←' KOMMT MAN WIEDER IN DEN"
160 PRINT:PRINT"TEXTMODUS ZURUECK."
170 PRINT:PRINT
180 INPUT "HINTERGRUND";HG
190 INPUT "FARBE 1";F1
200 INPUT "FARBE 2";F2
210 INPUT "FARBE 3";F3
220 INPUT "FILENAME";N$
230 :
240 COLOR 0,HG:COLOR 4,HG
250 COLOR 1,F1:COLOR 2,F2:COLOR 3,F3
260 GRAPHIC3,1:BLOAD (N$),ON B0,P(DEC("1C00")):BANK 15:SYS DEC("6B43"),16*(F1-1)
+(F2-1)
270 :
280 DO:GETKEY A$:LOOP UNTIL A$="←"
290 GRAPHIC 0:PRINT:PRINT"MIT 'GRAPHIC 3' WIRD DIE GRAFIK"
300 PRINT:PRINT"WIEDER ANGEZEIGT."
```

Dieses Programm haben wir zur Demonstration bereits eingesetzt, weshalb sich eine nähere Bedienungsanleitung erübrigt. Es sei nur darauf hingewiesen, daß die Fragen nach Farbcodes im Gegensatz zum »Apfelberge«-Programm beantwortet werden, da sie nicht voreingestellt sind.
Das Programm beruht im wesentlichen auf dem BLOAD-Befehl in Zeile 260 und der Warteschleife in Zeile 280, die den Linkspfeil abfragt.
Eine kleine Anregung: Programmieren Sie sich doch, wenn Sie viele Apfelberg-Grafiken erstellt haben, eine eigene Dia-Show! Dazu müssen Sie nur das Programm »LOAD APFELBERG« entsprechend abändern, daß es mehrere Bilder, deren Filenamen feststehen, hintereinander einliest.

Anwendung 2: Lissajous-Figuren

Ein weiterer Algorithmus zum Erzeugen von eindrucksvollen Grafiken soll auch in Basic 7.0 übertragen werden: die »Lissajous-Figuren«.
Der Name rührt vom französischen Physiker J. Lissajous her und bezeichnet Bahnkurven, die durch Überlagerung zweier ebener Schwingungen verschiedener Richtung zustande kommen. Diese sind kein theoretisches Gebilde, sondern physikalisch entdeckt worden und können mit dem Oszillographen sichtbar gemacht werden.
Folgendes Programm stellt neun Lissajous-Figuren dar:

Beschreibung: Lissajous-Figuren zeichnen
Filename: »lissajous-fig.«

```
100 REM ********************************
110 REM *                              *
120 REM *  PROGRAMM ZUM ZEICHNEN VON   *
130 REM *                              *
140 REM *       LISSAJOUS-FIGUREN      *
150 REM *       =================      *
160 REM *                              *
170 REM ********************************
180 :
190 REM *** KONSTANTEN SETZEN ***
200 REM *** ================= ***
210 :
220 X0=160:Y0=100:A=150:B=95:T0=π/180:REM BILDSCHIRMMITTE 160,100 IN X0,Y0
230 REM VERGROESSERUNGSFAKTOREN IN A,B
240 REM UMRECHNUNGSFAKTOR FUER GRADMASS IN BOGENMASS STEHT IN T0
250 :
260 REM *** GRAFIK-EINSTELLUNGEN ***
270 REM *** ==================== ***
280 :
290 GRAPHIC 1:REM *** HOCHAUFLOESENDE GRAFIK EINSCHALTEN
300 WIDTH 2:REM *** DOPPELTE PUNKTBREITE
310 COLOR0,8:COLOR4,8:COLOR1,1:REM FARBEN EINSTELLEN (SCHWARZ AUF GELB)
320 :
330 FOR I=1 TO 9:REM 9 GRAFIKEN
340 :SCNCLR:REM GRAFIK LOESCHEN
350 :READ D,N,M:REM WERTE AUS DATAZEILEN LESEN
360 :FOR W=0 TO 180*D STEP D
370 ::T=W*T0
380 ::X=X0+A*COS(N*T)
390 ::Y=Y0+B*SIN(M*T)
400 ::IF T THEN DRAW 1,X1,Y1 TO X,Y
410 ::X1=X:Y1=Y
420 :NEXT W
430 NEXT I
440 :
450 GRAPHIC 0:REM WIEDER AUF TEXTMODUS SCHALTEN
460 :
470 REM *************************
480 REM *                       *
490 REM *  DATAZEILEN FUER DIE  *
500 REM *                       *
510 REM *   UNTERSCHIEDLICHEN   *
520 REM *                       *
530 REM *       GRAFIKEN        *
540 REM *                       *
550 REM *************************
560 :
570 DATA 5,3,4  : REM FIGUR 1
580 DATA 77,1,2 : REM FIGUR 2
590 DATA 77,2,1 : REM FIGUR 3
600 DATA 77,2,3 : REM FIGUR 4
610 DATA 77,3,2 : REM FIGUR 5
620 DATA 94,1,1 : REM FIGUR 6
630 DATA 94,2,3 : REM FIGUR 7
640 DATA 94,1,2 : REM FIGUR 8
650 DATA 94,1,3 : REM FIGUR 9
```

Das Programm ist mit REM-Zeilen ausreichend kommentiert, was eine Beschreibung im Text überflüssig macht. Es soll nur noch erklärt werden, wie man weitere Grafiken anhängt:

- Ab Zeile 660 neue DATA-Zeilen (3 Elemente pro Grafik) anhängen
- Schleife in Zeile 330 (»TO 9«) an neue Anzahl der Grafiken anpassen

Als Demonstration, wie man erstaunliche Grafiken auch mit einer einzigen Basic-Zeile bewerkstelligen kann, mag folgendes Programm dienen:
Beschreibung: Lissajous-Figur mit einer einzigen Zeile zeichnen
Filename: »lissaj.einzeiler«

```
10 REM * LISSAJOUS-FIGUR ALS EINZEILER *
20 REM * ============================= *
30 :
40 GRAPHIC 1,1:COLOR 0,7:COLOR 4,7:COLOR 1,8:LOCATE 10,100:FOR K=0 TO 6.3 STEP .
01:DRAW TO 160-150*COS(3*K),100-75*SIN(5*K):NEXT
```

Anwendung 3: CIRCLE-Befehl vielseitig verwendet

Anhand von drei Programmen soll deutlich werden, wie viele Möglichkeiten der CIRCLE-Befehl bietet. Diese Programme lernen Sie am einfachsten kennen, wenn Sie sie von Diskette laden und starten und dann das Listing ansehen. Aufgrund der Kürze und Übersichtlichkeit der Programme erübrigen sich eingehendere Ausführungen.
Beschreibung: Anwendungsbeispiel 1 zum CIRCLE-Befehl
Filename: »circle-beisp. 1«

```
100 REM *******************************
110 REM *                             *
120 REM *   ANWENDUNGSBEISPIEL ZUM    *
130 REM *                             *
140 REM *   GRAFIKBEFEHL  "CIRCLE"    *
150 REM *                             *
160 REM *            (1)              *
170 REM *                             *
180 REM *******************************
190 :
200 GRAPHIC 0:INPUT "STEIGUNG";S:INPUT "FAKTOR";F:B=0
210 COLOR 0,7:COLOR 4,7:COLOR 1,2:GRAPHIC 1,1
220 FORA=200TO0STEP-3
230 :B=B+F
240 :CIRCLE,160,100,A,A,0,0,B,S
250 :NEXT
260 GETKEY A$
270 PRINT:RUN
```

Beschreibung: Anwendungsbeispiel 2 zum CIRCLE-Befehl
Filename: »circle-beisp. 2«

```
100 REM ****************************
110 REM *                          *
120 REM *  ANWENDUNGSBEISPIEL ZUM  *
130 REM *                          *
140 REM *  GRAFIKBEFEHL  "CIRCLE"  *
150 REM *                          *
160 REM *           (2)            *
170 REM *                          *
180 REM ****************************
190 :
200 COLOR4,15:COLOR1,1:COLOR0,8:GRAPHIC1,1:E=0
210 DO UNTIL E=180
220 :E=E+4
230 :CIRCLE 1,159,99,E,,,,E,72
240 LOOP
```

Beschreibung: Anwendungsbeispiel 3 zum CIRCLE-Befehl
Filename: »circle-beisp. 3«

```
100 REM ****************************
110 REM *                          *
120 REM *    ANWENDUNGSBEISPIEL    *
130 REM *                          *
140 REM * ZUM GRAFIKBEFEHL "CIRCLE" *
150 REM *                          *
160 REM *           (3)            *
170 REM *                          *
180 REM ****************************
190 :
200 COLOR 1,2:COLOR 0,6:COLOR 4,6:GRAPHIC 1,1:WIDTH 2
210 :
220 FOR X=5 TO 50 STEP 2
230 :CIRCLE,180+X*.8,315,100,90,45,X*1.2,X
240 :CIRCLE,180+X*.8,100,135,270,45,X*1.2,X
250 NEXT
260 :
270 DO:GETKEY A$:LOOP UNTIL A$="←"
280 GRAPHIC 0
```

Anwendung 4: Beispiel zum DRAW-Befehl

Auch zum DRAW-Befehl möchte ich Ihnen ein Beispiel geben:
Beschreibung: Anwendungsbeispiel zum DRAW-Befehl
Filename: »draw-beispiel«

```
100 REM ***************************
110 REM *                         *
120 REM *   ANWENDUNGSBEISPIEL    *
130 REM *                         *
140 REM *    ZUM GRAFIKBEFEHL     *
150 REM *                         *
160 REM *       "D R A W"         *
170 REM *                         *
180 REM ***************************
190 :
200 COLOR 1,2:COLOR 0,6:COLOR 4,6:GRAPHIC 1,1
210 K1=140/320:K2=100/240:K3=70/160
220 FOR X= 0 TO 320 STEP 8
230 :X1=X*.75:X2=X/2
240 :M1=K1*X
250 :M2=K2*X1
260 :M3=K3*X2
270 :DRAW ,320- X,M1 TO X,140
280 :DRAW ,260-X1,M2 TO X1+20,100
290 :DRAW ,200-X2,M3 TO 40+X2,70
300 NEXT
310 DO:GETKEY A$:LOOP UNTIL A$="←"
320 GRAPHIC 0
```

Die vorgestellten Anwendungsbeispiele sind Ergänzungen zu den Befehlserklärungen, die sicherlich den Einstieg in die Basic-7.0-Grafikprogrammierung erleichtern.
Wer sich näher für Grafik interessiert, findet weitere Informationen im schon so oft erwähnten Buch »Grafikprogrammierung C128« von Markt & Technik, Nummer MT 90202.

3.4.4 Befehlsgruppe »Sound«

Nicht nur die Grafik-, sondern auch die Soundeffekte müssen beim C64 über PEEK und POKE realisiert werden, weil Basic 2.0 keine entsprechenden Befehle hat. In Basic 7.0 kann man sich einiger Kommandos bedienen, die dies erleichtern.
Diese Befehle werden im C128-Handbuch in Kapitel 4.8 (Seiten 4-152 bis 4-166) ausführlich erklärt. Lesen Sie, wenn Sie sich für Soundprogrammierung interessieren, die dortigen Erklärungen durch und machen Sie dann in diesem Buch bei den Anwendungen weiter.
Zunächst sollen aber auch die ausgesprochenen »Sound-Muffel« ein paar Informationen bekommen, wie man auf einfache Weise in sogenannten »ernsthaften Programmen« Signaltöne programmiert. Dies interessiert Programmierer, die nur recht einfache Soundeffekte benötigen (z.B. als Warn- oder Signalton in einer Dateiverwaltung) und sich nicht erst durch seitenlange Ausführungen über technische Details schlagen wollen, erfahrungsgemäß am meisten, wird aber in der Literatur vernachlässigt.

Soundeffekte für Sound-Unkundige

Wenn Sie sich mit einen einfachen Signalton begnügen, den Sie in Anwendungsprogrammen oder ähnlichen verwenden wollen, können Sie den Befehl

```
PRINT CHR$(7);
```

verwenden, der in Kapitel 3.1 dieses Buches zusammen mit den anderen ASCII-Codes erläutert wird. Beachten Sie, daß dieser Signalton durch ESC-G und ESC-H ein- bzw. ausgeschaltet wird (siehe 2.1.2). Außerdem darf sich der Computer nicht im Quote Mode befinden, da sonst ein reverses G dargestellt wird und das Signal nicht ertönt.
In jedem Fall funktioniert folgender Befehl, der aber nicht so einfach zu merken ist:

```
BANK 15:SYS DEC("C98E")
```

In Maschinensprache: `JSR $C98E`

Dadurch wird unmittelbar die Routine des Basic-Interpreters, die für den Klingelton verantwortlich ist, aufgerufen.
Als Sirene bei Fehleingaben oder ähnlichen Anlässen eignet sich

```
SOUND 1,10000,1000,2,1000,500,2,2000
```

Wollen Sie ein wenig Abwechslung bei den Signaltönen, können Sie ab und zu auch folgenden Befehl verwenden:

```
SOUND 1,7493,60
```

Der letzte Wert (60) ist die Dauer des Tons in 1/60-Sekunden; der Beispielton hält also 60/60 = 1 Sekunde an. Natürlich darf dieser Wert verändert werden, solange der Wert im Bereich 1-32767 liegt.
Der zweite Wert (7493) ist die Frequenz des Tones, die ebenfalls variiert werden kann. Erlaubt sind Werte von 0 (tiefstmöglicher Ton) bis 65535 (höchstmöglicher Ton). Mit Hilfe der folgenden Eingabe können Sie eine bestimmte Frequenz suchen:

```
FOR I=0 TO 65535:SOUND 1,I,1:NEXT
```

Wenn diese Eingabe, die vom tiefsten bis zum höchsten Ton alle Frequenzen durchläuft, gerade eine Tonlage hat, deren Frequenz Sie erfahren wollen, unterbrechen Sie den C128 mit <RUN/STOP> und geben Sie »PRINT I« ein. Den auf diese Weise erhaltenen Wert können Sie als zweiten SOUND-Parameter einsetzen. <CONTROL>+<G> schaltet einen über SOUND erzeugten Ton vorzeitig ab.
Die Lautstärke eines Tones, der bei einer SOUND-Anweisung erklingt, kann mit einem einfach zu handhabenden Befehl reguliert werden:

```
VOL 0     Ausschalten der Tonausgabe
VOL 1     minimale Lautstärke
...
VOL 15    maximale Lautstärke
```

Zwischen 1 und 15 sind alle Werte erlaubt (je kleiner der Wert, desto weniger Lautstärke). Damit wollen wir es aber belassen, denn dieser kleine Abschnitt wendet sich an diejenigen, die nicht zu Soundprofis werden wollen, sondern Soundeffekte für den Hausgebrauch suchen.

Anwendungen der Soundbefehle

Wenn Sie das Kapitel 4.8 im Handbuch gelesen haben, so haben Sie folgende Befehle kennengelernt:

```
SOUND
VOL
FILTER
PLAY
TEMPO
ENVELOPE
```

Zusätzlich zu den Beispielen im Handbuch sollen noch zwei weitere Demoprogramme vorgestellt werden.
Beschreibung: Demo zu »TEMPO« und »PLAY«
Filename: »invention 13«

```
100 REM ******************************
110 REM *                            *
120 REM *  D E M O P R O G R A M M   *
130 REM *                            *
140 REM *  ZU DEN "SOUND"-BEFEHLEN   *
150 REM *                            *
160 REM *  TEMPO     UND       PLAY  *
170 REM *                            *
180 REM ******************************
190 :
200 SCNCLR
210 :
220 INPUT "GESCHWINDIGKEIT (0-255)";GS
230 TEMPO GS
240 :
250 DO
260 :READ A$
270 :PLAY A$
280 LOOP UNTIL A$="V201HA V103SAECE02QAM":REM NACH LETZTER DATA-ZEILE  SCHLEIFE
BEENDEN, UM "OUT OF DATA ERROR" ZU UMGEHEN
290 :
300 PRINT "NOCHMAL (J/N)";:DO:GETKEY A$:LOOP UNTIL A$="J" OR A$="N"
310 IF A$="J" THEN RUN:ELSE END
320 :
330 REM
340 REM *** DATEN FUER PLAY-BEFEHL ***
350 REM
360 DATA V104T7U8X0 V204T7U8X0
370 DATA V201IA V103IE V202QA V103SA04C03BEM
380 DATA V202I#G V103SB04D04IC V202SAEM
390 DATA V104IE V202SA03C V103I#G V202SBEM
400 DATA V104IE V202SB03DM
410 DATA V203IC V103SAE V202IA V103SA04CM
420 DATA V202I#G V103SBE V202IE V103SB04DM
430 DATA V104ICV 202SAE V103IA V202SA03CM
```

```
440 DATA V104QR V202SBEBO3DM
450 DATA V203IC V104SRE V202IA V104SCEM
460 DATA V203IC V103SA04C V202IA V102SEGM
470 DATA V103IF V203SD02A V103IA V202SFAM
480 DATA V104ID V202SDF V104IF V201SA02CM
490 DATA V201IB V104SFD V202ID V103SB04DM
500 DATA V202IG V103SGB V202IB V103SDFM
510 DATA V103IE V202SGE V103IG V202SEGM
520 DATA V104IC V202SCE V104IE V201SGBM
530 DATA V201IA V104SEC V202IC V103SA04CM
540 DATA V103IF V202SDF V104ID V201SB02DM
550 DATA V201IG V103SDB V201IB V103SGBM
560 DATA V103IE V202SCE V104IC V201SA02CM
570 DATA V201IF V104SC03A V201ID V103SFAM
580 DATA V103ID V201SG02G V103IB V202SFGM
590 DATA V201IA V104SC03A V202I#F V104SCEM
600 DATA V201IB V104SD03B V202I#G V104SDFM
610 DATA V202IC V104SEC V202IA V104SEGM
620 DATA V202ID V104SFE V202I$B V104SDCM
630 DATA V202I#G V103SB04C V202IF V104SDEM
640 DATA V202ID V104SFD V201IB V104S#GDM
650 DATA V202I#G V104SBD V202IA V104SCAM
660 DATA V202ID V104SFD V202IE V103SB04DM
670 DATA V202IF V103S#GB V202I#D V104SC03AM
680 DATA V202IE V103SEA V202IE V103SB#GM
690 DATA V201HA V103SAECE02QAM
```

Diese Routine können Sie auch in eigene Programme einbauen, da es sich als Titelmusik sehr gut eignet.
Beschreibung: Demo zu »VOL«, »ENVELOPE«, »TEMPO« und »PLAY«
Filename: »sound-demo«

```
100 REM ********************
110 REM *                  *
120 REM * BEISPIELPROGRAMM *
130 REM *                  *
140 REM *   FUER FOLGENDE  *
150 REM *                  *
160 REM * 'SOUND'-BEFEHLE: *
170 REM *                  *
180 REM *      "VOL",      *
190 REM *                  *
200 REM *   "ENVELOPE",    *
210 REM *                  *
220 REM * "TEMPO" & "PLAY" *
230 REM *                  *
240 REM ********************
250 :
260 VOL 15:REM LAUTSTAERKE
270 :
280 ENVELOPE 0,8,8,6,0,1:REM HUELLKURVE ETC.
290 :
300 TEMPO 30:REM GESCHWINDIGKEIT
310 :
320 PLAY"V1 T0 04 QCQDQEQFHGHG QAQA AQAHGR QAQAQAQAHGR QFQFQFQFHEHEQDQDQDQDHC" :
 REM MUSIKSTRING ABLAUFEN LASSEN
```

Dieses Programm spielt ein bekanntes Lied (probieren Sie's aus), das aber nicht für den Einbau in Ihre Programme vorgesehen ist...

Tip zur Soundprogrammierung

Wenn bei der Programmierung von Musikstücken ein Fehler auftritt, hat dies hin und wieder einen unangenehmen Dauerton zur Folge, da der letzte Ton unendlich lange gehalten wird. Dieser Dauerton kann, wie schon erwähnt, dadurch abgestellt werden, daß man

<CONTROL>+<G> (Klingelton)

drückt. Zeigt diese Tastenkombination nicht die gewünschte Wirkung, so gibt es folgende zwei Möglichkeiten:

- Der Dauerton liegt nicht auf Stimme 1. Dann hilft »VOL 0«.
- Diese Tastenkombination wurde vorher durch ESC-H verboten. In diesem Fall muß erst ESC-G (Klingelton erlauben) und dann <CONTROL>+<G> (Klingelton) gedrückt werden.

3.4.5 Befehlsgruppe »Ein-/Ausgabe«

Prinzipiell läßt sich jedes Programm in folgendes Schema einfügen:

- Eingabe der Daten
- Verarbeitung (Auswertung) der Daten
- Ausgabe der Daten

Der erste und dritte Schritt wird von speziellen Basic-7.0-Befehlen unterstützt; wir werden aber auch in 3.6 professionelle Ein-/Ausgabe-Techniken kennenlernen, die weit über die Leistungen der Basic-Befehle hinausgehen. Zunächst wollen wir aber die Standardbefehle des Basic 7.0 besprechen, die für Ein-/Ausgabe-Zwecke nützlich sind.

KEY – Der Schlüssel zur Batch-Verarbeitung

Den Befehl »KEY« zur Definition einer eigenen Funktionstastenbelegung haben wir bereits in 2.1.1 besprochen. Dabei wurde noch nicht erwähnt, daß der KEY-Befehl auch in Programmen großen Nutzen bringt. Durch die Belegung einer Funktionstaste mit bestimmten Texten, die bei Eingaben innerhalb des Programms dem Anwender die Programmbedienung erleichtern, kann der Bedienungskomfort erheblich gesteigert werden. So können mehrere Tastendrücke durch einen einzigen ersetzt werden. In einer Textverarbeitung wäre es denkbar, bestimmte immer wiederkehrende Redewendungen auf die Funktionstasten zu legen.
Die Abfrageroutine des Programms empfängt die einzelnen Zeichen der Funktionstastenbelegung auf die gleiche Weise, wie wenn die einzelnen Tasten hintereinander gedrückt worden wären.
Folgendes Programm (bitte eintippen!) zeigt dies:

```
10 KEY 1,"DIESER TEXT LIEGT AUF F1!"
20 PRINT "BITTE F1 DRUECKEN"
30 DO:GETKEY A$:PRINT A$;:LOOP
```

Eine andere Möglichkeit wäre folgende, die statt einer GETKEY-Abfrage den INPUT-Befehl anspricht:

```
10 KEY 1,"DIESER TEXT LIEGT AUF F1!"+CHR$(13):REM + <RETURN>
20 PRINT "BITTE <F1> DRUECKEN"
30 INPUT A$:PRINT A$
```

Eine Eingabetechnik, die hauptsächlich auf größeren Computern (Personal-Computer), aber auch im CP/M-Modus des C128 realisiert wird, ist mit einem kleinen Trick auch auf dem C128 unter Basic 7.0 möglich: die »Batch-Verarbeitung« (Stapelverarbeitung).
Damit ist gemeint, daß dem Computer anstatt einer einzigen Eingabe, die mit <RETURN> quittiert wird, gleich mehrere Eingaben mitgeteilt werden, die er dann für den Bedarfsfall (INPUT oder GET-Befehl oder Direkteingabemodus des Basic-Interpreters) aufhebt. Dies würde z.B. bedeuten, daß nicht nur der Ladebefehl eines Programms angegeben wird, sondern zugleich die Antworten auf spätere INPUT-Eingaben des Programms. Nehmen wir als Beispiel das Programm »LOAD APFELBERG«, das in Abschnitt 3.4.3 beschrieben wird. Dieses erfordert nach dem Laden und Starten die Eingabe von vier Farben und einem Filenamen. In der Regel gibt man alles getrennt ein, wenn man durch den Cursor und/oder entsprechende Texte am Bildschirm dazu aufgefordert wird:

1. Ladebefehl: RUN "LOAD APFELBERG" <RETURN>
2. Hintergrundfarbe: 1 <RETURN>
3. Farbe 1: 2 <RETURN>
4. Farbe 2: 3 <RETURN>
5. Farbe 3: 7 <RETURN>
6. Filename: APFELBERG.PIC <RETURN>

Sicher wäre es bequemer, alle Eingaben auf einmal zu machen:

```
RUN "LOAD APFELBERG" <RETURN> 1 <RETURN> 2 <RETURN> 3 <RETURN> 7
<RETURN> APFELBERG.PIC <RETURN>
```

Dies ist jedoch nicht möglich, da der Tastaturpuffer (Bereich, in dem Tastatureingaben zwischengespeichert werden) des C128 nicht groß genug ist und jedes <RETURN> eine direkte Ausführung eines Befehls veranlaßt. Ansonsten könnte man zwar keinen Tastendruck sparen, aber man würde den Computer längere Zeit laufen lassen können, ohne daß neue Eingaben gefordert werden; zudem müßte man sich nicht darum kümmern, wieviel Zeit zwischen der einen und der nächsten Eingabe vergeht (etwa weil das Programm erst geladen werden muß, bevor neue Eingaben angenommen werden).
Am Beispiel von »LOAD APFELBERG« wollen wir dies mit Hilfe der programmierbaren Funktionstastenbelegung ausprobieren. Wir legen alle Tastendrücke - einschließlich RETURNs - auf die Taste F1:

```
KEY1,"RUN"+CHR$(34)+"LOAD APFELBERG"+CHR$(34)+CHR$(13)+
"1"+CHR$(13)+"2"+CHR$(13)+"3"+CHR$(13)+"7"+CHR$(13)+
"APFELBERG.PIC"+CHR$(13)
```

und betätigen <F1>. Sehen Sie selbst, was passiert.

Dieses Beispiel mag vielleicht noch nicht den vollen Nutzen der Batch-Verarbeitung zeigen, aber zumindest haben Sie jetzt einen Einblick bekommen. Außerdem muß man bedenken, daß der C128 eigentlich nicht batch-fähig ist; die Lösung mit KEY ist nur ein Notbehelf.
Sie sollten ferner berücksichtigen, daß dies innerhalb eines Programms sehr nützlich sein kann, wenn mehrere Eingaben durch einen Tastendruck erledigt werden. Beispielsweise könnte das Programm »Load Apfelberg« am Anfang die Eingabe der Standardwerte für Farben und Filenamen auf <F1> legen:

```
KEY 1,"1"+CHR$(13)+"2"+CHR$(13)+"3"+CHR$(13)+"7"+CHR$(13)+
"APFELBERG.PIC"+CHR$(13)
```

Übrigens: Der Gegensatz zur Batch-Verarbeitung ist die normale Eingabeform von C64/C128 und trägt die Bezeichnung »Dialog-Verarbeitung«: es kann nur jeweils eine Eingabe getätigt werden, wenn man dazu aufgefordert wird (die Batch-Verarbeitung speichert mehrere Eingaben und läßt diese erst bei Bedarf wirksam werden).
Dies wird als Eingabe-Dialog bezeichnet.

CHAR - Textausgabe an beliebige Position (PRINT AT)

Die Positionierung des Cursors über die Cursor-Steuerzeichen (CRSR RIGHT, CRSR LEFT, CRSR UP, CRSR DOWN) ist

- umständlich
- unübersichtlich
- langsam
- fehlerträchtig bei der Programmentwicklung

Deshalb sollte die Ausgabe eines Textes an eine beliebige Bildschirmposition mit dem CHAR-Befehl (siehe Handbuch, Seite 4-28f) programmiert werden, der nicht nur im Grafikmodus arbeitet:

```
CHAR,Spalte (0-39 bzw. 0-79),Zeile (0-24),"Text in Anführungszeichen"
```

Hängt man »,1« an, wird der Text revers ausgegeben.
Geben Sie im Textmodus (40 oder 80 Zeichen pro Zeile) folgendes Beispiel ein:

```
CHAR,10,20,"TEXT IN SPALTE 10/ZEILE 20",1
```

Wenn Sie das »,1« weglassen, wird der Text nicht revers gedruckt.
Wichtig ist, daß der Textstring nicht wie bei PRINT angegeben wird; Semikolon, Komma, TAB, SPC usw. sind bei CHAR unzulässig. Als Faustregel können Sie sich merken, daß der Textstring dann CHAR-tauglich ist, wenn er hinter

```
A$=             (oder "LET A$=")
```

stehen könnte. Dies läßt sich in Zweifelsfällen im Direktmodus prüfen (fehlerhafte Eingabe ergibt »?SYNTAX ERROR«).

Wenn Zahlenwerte ausgegeben werden sollen, müssen diese über STR$ in Strings umgewandelt werden:

```
PRINT 5          aber CHAR,10,10,STR$(5)
PRINT "ZAHL";Z aber CHAR,10,10,"ZAHL:"+STR$(Z)
```

Am zweiten Beispiel sieht man auch, daß das Semikolon durch »+« ersetzt werden muß. Die Funktion des Kommas hinter PRINT kann meist durch CHR$(9) ausgeübt werden:

```
PRINT "ZAHL:",5
```

entspricht also

```
CHAR,10,10,"ZAHL:"+CHR$(9)+STR$(5)
```

in etwa, allerdings nicht vollständig: Das Komma ist ein Zehner-, <TAB> ein Achtertabulator.

Der CHAR-Befehl wurde schon im ersten Beispielprogramm dieses Buches eingesetzt (Abschnitt 2.1.3, Programm »'window'-demo«), aber dort noch nicht erklärt. Jetzt können Sie dieses Programm von Anfang bis Ende verstehen!
Zur Funktionsweise des CHAR-Befehls im Textmodus. Zunächst wird dursor auf die angegebene Position gesetzt, dann wird der darauffolgende String ähnlich wie über PRINT ausgegeben. Gegebenenfalls wird dabei ein angehängtes »,1« berücksichtigt (reverse Anzeige). Steuerzeichen in der Zeichenkette werden wie bei PRINT ausgeführt.
Wenn man den auszugebenden Text korrekt angibt, also nicht wie bei PRINT, dürfte es keine Probleme geben.
Ein PRINT-Befehl richtet sich übrigens, was die Cursorposition anbelangt, nach einem vorausgegangenen CHAR:

```
CHAR,10,10,"CHAR-TEXT!":PRINT "PRINT-TEXT"
```

zeigt dies.

COLOR – Komfortable Farbeinstellung in Basic 7.0

Der Befehl COLOR kann auch im Textmodus verwendet werden:

COLOR 0,Farbcode	Hintergrund (40-Zeichen-Modus)
COLOR 4,Farbcode	Rahmenfarbe (40-Zeichen-Modus)
COLOR 5,Farbcode	Textfarbe (40- und 80-Zeichen-Modus)
COLOR 6,Farbcode	Rahmen und Hintergrund (80-Zeichen-Modus)

Die Werte 1-3 führen zwar zu keiner Fehlermeldung, sind aber nur in Verbindung mit Grafik sinnvoll (siehe 3.4.3).

SCNCLR – Ersatz für PRINT CHR$(147)

Der Befehl SCNCLR (SCREEN CLEAR = Bildschirm löschen) kann, wie schon in 3.4.3 beiläufig gesagt, auch Textbildschirme löschen:

SCNCLR(0)	löscht den 40-Zeichen-Bildschirm,
SCNCLR(5)	löscht den 80-Zeichen-Bildschirm.
SCNCLR	löscht den aktuellen Bildschirm (im Textmodus also den aktuellen Textbildschirm)

Die letzte Variante ist, solange nicht die hochauflösende Grafik eingeschaltet ist, ein vollwertiger Ersatz für »PRINT CHR$(147);«. Deshalb sollte man bei der Programmierung ohne das etwas undurchsichtige Steuerzeichen CHR$(147) auskommen.

GRAPHIC – Auswahl des Textmodus

Der Befehl GRAPHIC dient nicht nur zum Einschalten der hochauflösenden Grafik, sondern auch – wie wir schon wissen – der Auswahl des Textmodus:

GRAPHIC 0	Textmodus am 40-Zeichen-Bildschirm
GRAPHIC 5	Textmodus am 80-Zeichen-Bildschirm

Dies ist bei Programmen, die nur mit einer bestimmten Zeichenbreite einwandfrei funktionieren, die beste Möglichkeit, um den Textmodus festzulegen. Zusätzlich sollte ein anwenderfreundliches Programm wie die Programme in diesem Buch eine Meldung ausgeben, wenn ursprünglich ein anderer Modus eingeschaltet ist, damit der Benutzer nicht meint, das Programm sei »abgestürzt«.

GETKEY – Sinnvolle Erweiterung des GET-Befehls

Der Befehl »GETKEY A$«, der auch als »GET KEY A$« geschrieben werden kann, ist schnell erklärt. Er entspricht der in Basic 2.0 nötigen Zeile

```
10 GET A$:IF A$<>"" THEN 10
```

GETKEY wartet also erst, bis eine Taste gedrückt wird, und arbeitet dann weiter wie der normale GET-Befehl.

JOY – Komfortable Abfrage der Joysticks

Dieser Befehl zur Joystick-Abfrage kann auch mit einer »Maus« (neuartiges Eingabegerät, welches vor allem durch die Computer Apple MacIntosh, Commodore AMIGA und Atari ST bekannt ist) verwendet werden, sofern diese an den C128 angeschlossen werden kann und »joystick-kompatibel« ist.
Für Spieleprogrammierung sicher häufig benötigt, liefert

JOY(1)	den Zustand von Joystick-Port 1,
JOY(2)	den Zustand von Joystick-Port 2.

Dabei sind folgende Werte als Ergebnis möglich:

0	keine Joystickbewegung
1	nach oben (Norden)
2	nach links oben (Nordwest)

3	nach links (Westen)
4	nach links unten (Südwest)
5	nach unten (Süd)
6	nach rechts unten (Südost)
7	nach rechts (Osten)
8	nach rechts oben (Nordost)

Bei gleichzeitigem Drücken des Feuerknopfes erhöht sich der Wert, den die JOY-Funktion zurückgibt, um 128:

128 (128+0)	Feuerknopf, aber keine Joystickbewegung
129 (128+1)	Feuerknopf + oben
130 (128+2)	Feuerknopf + links oben

usw. (siehe Bild 3.2).

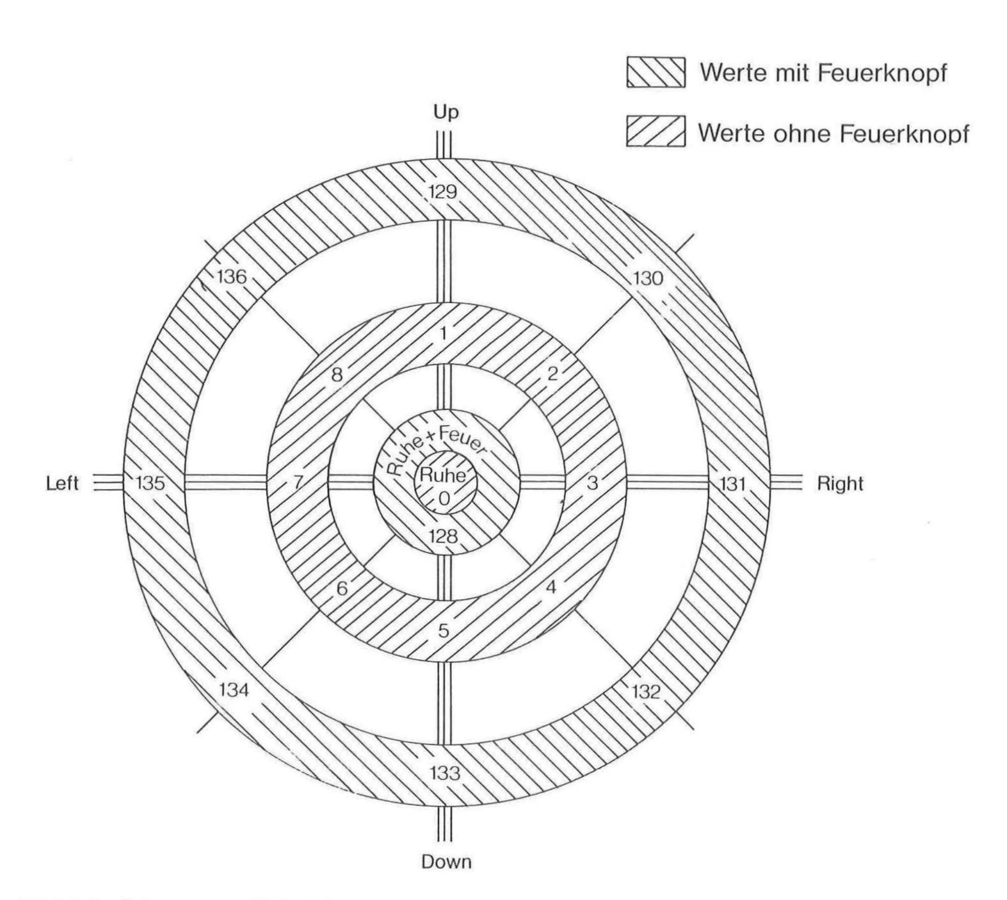


Bild 3.3: *Schema zur JOY-Abfrage*

Soll der Feuerknopf ignoriert werden, schreibt man

JOY(1) AND 127 statt JOY(1)
JOY(2) AND 127 statt JOY(2)

Folgende DO-LOOP-Schleife wartet hingegen, bis der Feuerknopf ausgelöst wird:

```
DO:LOOP UNTIL JOY(n)>127
```

Dabei muß für »n« die Nummer des Joystick-Ports eingesetzt werden.

Übrigens: Die Abfrage der Joysticks mit dem C64, die im C64-Handbuch nicht beschrieben wird, wird im C128-Handbuch auf Seite G-1 behandelt (für den C64-Modus).

POT - Drehregler (Paddles) abfragen

Die Funktion

```
POT(n)
```

liefert den Wert eines von 4 angeschlossenen Paddles (Drehregler). Für »n« sind Werte von 1 bis 4 erlaubt, die die Nummer des angewählten Drehreglers angeben. Die POT-Funktion liefert je nach Stellung des Drehknopfs Werte zwischen 0 und 255. Werte über 255 bedeuten, daß zusätzlich der Feuerknopf gedrückt wurde. Soll der Feuerknopf ignoriert werden, schreibt man

POT(n) AND 255 statt POT(n)

Folgende DO-LOOP-Schleife wartet hingegen auf das Auslösen des Feuerknopfes:

```
DO:LOOP UNTIL POT(n) >255
```

»n« muß dabei natürlich durch eine Zahl (1-4) ersetzt werden.
Die Drehregler-Abfrage mit dem C64 (oder C64-Modus) wird auf Seite G-1 des C128-Handbuches erläutert.

PEN - Lichtgriffel (Lightpen) abfragen

Da nur eine kleine Minderheit unter den C128-Anwendern über einen Lichtgriffel (englisch: Lightpen) verfügt, ist dieser Befehl nicht von allgemeinem Interesse. Wer einen Lichtgriffel hat, kann im C128-Handbuch auf Seite 4-82 nachlesen.
Gegebenenfalls ist auch die Anleitung zu Ihrem Lichtgriffel zu berücksichtigen, was z.B. das Problem anbelangt, daß der Bildschirm weiß sein muß. Dieses Erfordernis ist nämlich von Produkt zu Produkt unterschiedlich, um den Lightpen betreiben zu können.

PRINT USING - Formatierte Zahlen- und Textausgabe

Dieser Befehl gibt eine Liste von Ausdrücken auf dem Bildschirm oder einem Ausgabegerät formatiert aus. Dazu wird ein Formatstring angegeben, der alle Informationen über das Aussehen des Textes im Ausdruck (Druckfeld) enthalten muß. Beispiel:

```
PRINT USING "######";"TEXT"
```

Damit haben wir schon unser erstes Formatzeichen kennengelernt (»#«).
Jedes »#« im Formatstring steht für ein Zeichen im Ausdruck. In jedem Formatstring muß mindestens ein »#« vorhanden sein. Hat ein auszugebender String mehr Zeichen als Druckstellen mit »#« reserviert werden, wird der String gekürzt (auf die richtige Länge gestutzt), ansonsten mit Leerzeichen aufgefüllt. Dabei werden die Daten linksbündig gedruckt, d.h. eventuelle Leerzeichen werden rechts angehängt, damit links im Druckfeld (so heißt das Ausgabefeld) nach Möglichkeit immer ein sichtbares Zeichen steht.
Sollen die Daten nicht linksbündig, sondern rechtsbündig im Druckfeld stehen, muß im Formatstring anstelle eines »#« ein »>« stehen:

```
PRINT USING ">#####";"TEXT"
```

Für die Zentrierung im Druckfeld ist »=« anstelle eines »#« zu verwenden:

```
PRINT USING "=#####";"TEXT"
```

Die bisherigen Formatsymbole bezogen sich auf Zeichenketten-Druckfelder, also auf die Ausgabe von Strings mittels PRINT USING.
Es gibt aber auch numerische Druckfelder (zur Ausgabe von Zahlen), wofür folgende Formatzeichen zur Verfügung stehen:

- \# Wie bei Strings, nur, daß Sterne (*) bei zu hohen Zahlen das Druckfeld auffüllen.
- \+ Muß an erster oder letzter Position im Formatstring stehen, damit an dessen Stelle das Vorzeichen der auszugebenden Zahl erscheint.
- \- Siehe »+«.
- . Legt die Dezimalpunkt-Position fest. Je Formatstring ist (sinnvollerweise) nur ein Dezimalpunkt möglich.
- , Zur besseren Lesbarkeit von großen Zahlen wird alle 3 Stellen ein Komma gesetzt. Das Komma im Formatstring legt diese Position fest. Diesem muß im Formatstring mindestens ein # vorangestellt sein.
- $ Steht in der Formatkette hinter mindestens einem # ein $, so wird vor die ganze auszugebende Zahl (also vor die erste gültige Ziffer) ein Dollar-Zeichen gesetzt.

Die Exponentialdarstellung wird erzwungen, wenn der Formatstring mit vier »↑« endet (↑↑↑↑).
Diesen muß unbedingt ein Plus- oder Minus-Zeichen (»+« oder »-«) folgen.
Die genaue Syntax entnehmen Sie bitte dem C128-Handbuch, Seite 4-88.
Unzählige Beispiele finden Sie dort auf den Seiten 4-90 und 4-91. Das Ausprobieren einiger Beispiele ist sehr verständnisfördernd, da der PRINT USING-Befehl sehr vielfältig zu gebrauchen ist. Allerdings wird eine Variante merkwürdigerweise nicht erwähnt: der Formatstring kann auch gewöhnlichen Text beinhalten; dieser wird dann wie bei PRINT ausgegeben, darf aber logischerweise keine Formatsteuerzeichen enthalten, da diese sonst ausgeführt würden:

```
PRINT USING "BETRAG: ####.##"; 55.45, 575.03, 3.14
```

PUDEF – Symbole für PRINT USING umdefinieren

Nicht nur, weil die Entwickler des Basic 7.0 Amerikaner waren, haben sie die bei Computern übliche amerikanische Zahlenschreibweise verwendet, welche sich von der deutschen in einigen Punkten unterscheidet, obwohl die Ziffern die gleichen sind:
Amerikanisch: 3.14 Deutsch: 3,14 (in Worten: drei Komma vierzehn)
Amerikanisch: 3,000 Deutsch: 3.000 (in Worten: dreitausend)

Die wesentlichen Änderungen sind also, daß
a) die Amerikaner anstelle des Kommas den Dezimalpunkt verwenden, wie wir ihn von der Basic-Syntax kennen, und daß
b) wir im deutschen Sprachraum zur Gliederung größerer Zahlen nicht das Komma, sondern den Punkt (oder ein Leerzeichen) nehmen.

Mit dem PUDEF-Befehl ist es aber möglich, dem PRINT USING-Befehl die deutsche Zahlenschreibweise beizubringen:

```
PUDEF " .,"
```

Die Syntax von PUDEF und die Funktionsweise entnehmen Sie bitte dem C128-Handbuch, Seite 4-92. Beispiel 2 auf Seite 4-93 ist jedoch falsch, da im PUDEF-String das Leerzeichen an erster Stelle fehlt:
Statt

```
10 PUDEF".,"
```

muß es

```
10 PUDEF" .,"
```

heißen. Die Befehlserklärung ist allerdings einwandfrei (sonst würde ich nicht darauf verweisen, sondern eine eigene Erklärung geben).

WINDOW – Window definieren

Der Befehl WINDOW zur Definition eines Windows wurde bereits in 2.1.3 vorgestellt, wo die Window-Technik ausführlich an zahlreichen Beispielen erklärt wird. Dort finden Sie auch ein speziell auf den WINDOW-Befehl zugeschnittenes Demoprogramm.

RWINDOW – Informationen über Window bzw. Bildschirm holen

Die Funktion

```
RWINDOW(n)
```

trifft, abhängig vom Wert »n« (0,1 oder 2), eine Aussage über den aktuellen Bildschirm bzw. das aktuelle Window:

RWINDOW(0)	liefert die Nummer der untersten Zeile im Window (0-24).
RWINDOW(1)	liefert die Nummer der am weitesten rechts stehenden Spalte im Window (0-39 im 40-,0-79 im 80-Zeichen-Modus).
RWINDOW(2)	liefert den Textmodus, siehe 2.2.2.

Leider können auf diese Weise nicht die oberste Zeile und am weitesten links liegende Spalte ermittelt werden; dafür benötigt man die PEEK-Funktion:

PEEK(229)	liefert die Nummer der obersten Zeile (0-24).
PEEK(230)	liefert die nummer der am weitesten links stehenden Spalte im Window (0-39 bzw. 0-79).

Auch RWINDOW(0) und RWINDOW(1) können mit PEEK umschrieben werden, was vor allem für Maschinenprogrammierer wichtig ist, weil sie dann die RWINDOW-Funktion leichter in Assembler schreiben können:

PEEK(228)	entspricht RWINDOW(0)
PEEK(231)	entspricht RWINDOW(1)

Wie man den RWINDOW(2)-Befehl durch PEEK ersetzt, finden Sie in 2.2.2.

Diskettenbefehle des Basic 7.0

Basic 7.0 verfügt über eine Reihe von Befehlen, die die Programmierung einer angeschlossenen Diskettenstation erheblich vereinfachen. Diese Befehle sind in Kapitel 6 des C128-Handbuches ausführlich beschrieben, das Sie jetzt bitte lesen, sofern Sie es nicht ohnehin schon getan haben.
Dort und im Anleitungsbuch zur Floppy 1570/71 hat sich jedoch ein Fehler bei der Erläuterung des RECORD-Befehls eingeschlichen. Es heißt, daß die Angabe der Byteposition hinter der Datensatznummer optional, das heißt nicht unbedingt notwendig, ist. Dies ist schlicht und einfach verkehrt. Bei einem RECORD-Befehl muß immer die Bytenummer angegeben werden, auf die in einem Datensatz positioniert werden soll.
Im Floppy-1570/71-Handbuch von Commodore fällt Ihnen vielleicht folgender Satz ins Auge: *»Sicherheitsmaßnahme: Jeder Record# -Befehl muß zweimal angegeben werden«.* Um die höchst unwahrscheinliche Möglichkeit der Zerstörung relativer Dateidaten auszuschließen, müssen die RECORD#-Befehle zweimal angegeben werden, bevor ein Datensatz gelesen wird.
Dieser Fehler, der offenbar Commodore selbst so mysteriös ist, daß eine Warnung an den Benutzer erfolgt, anstatt den Fehler aus dem Gerät zu entfernen, ist in Wirklichkeit nicht vorhanden. Hier sind die Hersteller auf einen Fehler im eigenen Bedienungshandbuch hereingefallen. Die »Möglichkeit der Zerstörung relativer Dateidaten« tritt nämlich nur dann auf, wenn im RECORD#-Befehl die Byteangabe, die angeblich optional ist, weggelassen wird. In diesem Fall kann es tatsächlich vorkommen, daß die Floppy vom Computer empfangene Daten ins Leere schreibt, wobei sie dann zwangsläufig verlorengehen.
Und selbst, wenn ein solcher Fehler vorhanden wäre (was er aber nicht ist), könnte die zweimalige Angabe des RECORD#-Befehls auch nicht helfen.
Geben Sie also immer die Bytezahl bei der Positionierung an, und Sie werden sehen, daß alles einwandfrei funktioniert, auch wenn der RECORD#-Befehl nur einmal angegeben wird.
Weitere Informationen zur Programmierung der Floppy finden Sie im Kapitel 7 dieses Buches.

3.4.6 Befehlsgruppe »Fehlerbehandlung und sonstige«

Dies ist die letzte Befehlsgruppe, die wir besprechen wollen, bevor wir uns mit dem Umschreiben von C64-Programmen und Basic-7.0-Anwendungen sowie Erweiterungen des Basic 7.0 auseinandersetzen.

TRAP – Programmierte Fehlerbehandlung

Der Befehl TRAP, der in ähnlicher Form als »ON ERROR GOTO« (in Simon's Basic: »ON ERROR:GOTO«) in Basic-Erweiterungen zum C64 bekannt wurde, ermöglicht es, auf Basic-Fehlermeldungen programmgesteuert zu reagieren. Das Programm wird also nicht mehr abgebrochen, sondern in einem bestimmten Programmteil fortgesetzt, der den Fehler eventuell beheben kann oder zumindest einen Programmabbruch verhindert.
Dazu muß mit

```
TRAP Zeilennummer
```

die Zeile mitgeteilt werden, ab der die Fehlerbehandlungsroutine beginnt. Beispiel:

```
100 TRAP 10000:REM Eigene Fehlerbehandlungsroutine ab Zeile 10000
110 XXXXXXX:REM Ergibt "?SYNTAX ERROR"
...
...
10000 PRINT "FEHLER ABGEFANGEN!"
10010 HELP
```

Durch Weglassen der Zeilennummer wird eine vorher aktivierte Fehlerunterbrechung ausgeschaltet.
Tritt nach einem TRAP-Befehl (mit Angabe der Zeilennummer) ein Fehler auf, so wird in die definierte Fehlerbehandlungsroutine verzweigt. Dies gilt für alle Fehlerbedingungen einschließlich der <RUN/STOP>-Taste. Wenn sich allerdings ein Fehler in der Fehlerbehandlungsroutine selbst befindet, wird eine Fehlermeldung ausgegeben.
Beim Auftreten eines Fehlers außerhalb der Fehlerbehandlungsroutine wird die fehlerhafte Stelle gemerkt und die Fehlerbearbeitung des Programms ausgeführt.
Dann steht in der Variablen »EL« (Abkürzung für »error line«) die Nummer der Zeile, in der der Fehler auftrat, und in »ER« der Fehlercode. Folgende Fehlercodes gibt es für die Variable ER (Abkürzung für »error«), die auch im Direktmodus über »PRINT ER« geprüft werden kann (Tabelle 3.3).

Tabelle 3.3: *Werte der Systemvariablen ER (Fehlercodes)*

1	TOO MANY FILES	
2	FILE OPEN	
3	FILE NOT OPEN	
4	FILE NOT FOUND	
5	DEVICE NOT PRESENT	
6	NOT INPUT FILE	
7	NOT OUTPUT FILE	
8	MISSING FILE NAME	
9	ILLEGAL DEVICE NUMBER	
10	NEXT WITHOUT FOR	
11	SYNTAX	
12	RETURN WITHOUT GOSUB	
13	OUT OF DATA	
14	ILLEGAL QUANTITY	
15	OVERFLOW	
16	OUT OF MEMORY	
17	UNDEF'D STATEMENT	
18	BAD SUBSCRIPT	
19	REDIM'D ARRAY	
20	DIVISION BY ZERO	
21	ILLEGAL DIRECT	
22	TYPE MISMATCH	
23	STRING TOO LONG	
24	FILE DATA	
25	FORMULA TOO COMPLEX	
26	CAN'T CONTINUE	
27	UNDEF'D FUNCTION	
28	VERIFY	
29	LOAD	
30	BREAK	
31	CAN'T RESUME	
32	LOOP NOT FOUND	
33	LOOP WITHOUT DO	
34	DIRECT MODE ONLY	
35	NO GRAPHICS AREA	
36	BAD DISK	
37	BEND NOT FOUND	(siehe 3.4.2, BEGIN/BEND)
38	LINE NUMBER TOO LARGE	(siehe 3.4.1, RENUMBER)
39	UNRESOLVED REFERENCE	(siehe 3.4.1, RENUMBER)
40	UNIMPLEMENTED COMMAND	(siehe 3.4.6, QUIT/OFF)
41	FILE READ	(siehe 7.3, Boot-Vorgang)

Die Meldungen 1-30 sind alte Bekannte vom C64.
Diese sind, zusammen mit den C128-spezifischen Fehlern 31-36, in Kapitel 8 des C128-Handbuches beschrieben. Da dort die Fehlermeldungen 37-41 nicht erwähnt werden, ist in der obigen Übersicht zusätzlich angegeben, wo in diesem Buch eine Fehlerbeschreibung steht.
Den Text zu einer Fehlermeldung kann man auf einfache Weise mit

```
PRINT ERR$(Fehlernummer)
```

ausgeben lassen. »PRINT ERR$(5)« führt beispielsweise zur Anzeige von »DEVICE NOT PRESENT«. Im Gegensatz zu einer »echten« Fehlermeldung wird dabei aber das Programm nicht abgebrochen.

HELP – Auch bei der Fehlerbehandlung hilfreich

In der Fehlerbehandlungsroutine darf auch der HELP-Befehl, der in 3.4.1 erklärt wurde, stehen. Dies ist sogar vorgesehen, denn HELP darf auch in Programmen ohne Einschränkung verwendet werden.

RESUME – Rücksprung nach Fehlerbehandlung

Die Fehlerbehandlungsroutine wird also von den Variablen EL und ER informiert. Wenn dann auf den Fehler reagiert wurde, kann ein Rücksprung mit dem RESUME-Befehl ausgelöst werden. Dabei stehen drei Varianten zur Verfügung:

```
RESUME
```

setzt das Programm ab der fehlerhaften Anweisung fort. Dies ist dann sinnvoll, wenn die Fehlerursache (z.B. ein nicht angeschaltetes Gerät) behoben werden konnte und der Befehl im zweiten Anlauf korrekt ausgeführt werden kann.
Zweite RESUME-Möglichkeit

```
RESUME NEXT
```

setzt das Programm unmittelbar nach dem Befehl, der den Fehler ausgelöst hat, fort.
Dritte und letzte mögliche Form der Syntax:

```
RESUME Zeilennummer
```

bewirkt ein Fortsetzen des Programms ab der durch den Parameter »Zeilennummer« definierten Programmzeile.
Ein Beispiel dafür finden Sie im C128-Handbuch auf Seite 4-101.

DEC und HEX$ – Der C128 versteht auch Hexadezimalzahlen

Das üblicherweise in Basic verwendete Zahlensystem ist das Dezimalsystem (Zehnersystem), welches wir auch im alltäglichen Gebrauch einsetzen. Es gibt aber auch andere Zahlenschreibweisen, von denen das Hexadezimalsystem (16er-System) bei der Programmierung in Maschinensprache das wichtigste ist. Für Basic-Programmierer ist es von geringem Nutzen; deshalb

sollen Sie hier nicht mit Erklärungen zum Hexadezimalsystem belastet werden, mit denen Sie danach nichts anfangen können. Für diejenigen, die aber dieses System beherrschen (z.B. Maschinensprachler), werden nur die Befehle DEC und HEX$ erläutert.
Zur Umwandlung einer Dezimalzahl in die Hexadezimalschreibweise dient die Funktion

```
HEX$(n)
```

»n« ist dabei eine Dezimalzahl im Bereich von 0 bis 65535, wodurch 64K Adreßraum (= 1 Bank) erfaßt werden.
Das Ergebnis der HEX$-Funktion ist, wie schon das Dollarzeichen in »HEX$« sagt, ein String, der aus exakt 4 Stellen besteht; ggf. wird mit Nullen aufgefüllt, wie es Maschinensprache-Monitore auch handhaben. Es ist auch klar einzusehen, warum Hexadezimalzahlen vom C128 nur als Strings verarbeitet werden können: die Ziffern A-F sind im Dezimalsystem, welches in Basic üblich ist, nicht numerisch und dürfen demzufolge nur in Zeichenketten stehen, da diese auch Buchstaben einschließen können.
Die umgekehrte Umrechnung erledigt

```
DEC("Hexadezimalzahl")
```

Das Ergebnis ist hier eine Dezimalzahl; als Parameter benötigt DEC aus den bei HEX$ genannten Gründen einen String. Die durch die Zeichenkette angegebene Hexadezimalzahl muß im Bereich von $0000 bis $FFFF liegen; führende Nullen, um die Zeichenkette auf 4 Zeichen zu erweitern, sind nicht nötig, und das assembler-übliche Dollarzeichen muß weggelassen werden.

Beispiele für HEX$ und DEC:

PRINT HEX$(49152)	Ergebnis: C000
PRINT HEX$(1039)	Ergebnis: 040F
PRINT HEX$(50)	Ergebnis: 0032
PRINT HEX$(11)	Ergebnis: 000B
PRINT HEX$(0)	Ergebnis: 0000
PRINT DEC(»C000«)	Ergebnis: 49152
PRINT DEC(»40F«)	Ergebnis: 1039
PRINT DEC(»32«)	Ergebnis: 50
PRINT DEC(»B«)	Ergebnis: 11
PRINT DEC(»0«)	Ergebnis: 0

Für den Fall, daß Ihnen diese Beispiele nicht ausreichen sollten:
Ein Beispielprogramm, in dem die Befehle DEC und HEX$ im Mittelpunkt stehen, finden Sie unter den Demoprogrammen für »Label 128« in 3.4.2:

```
"LABEL 128.DEMO 5", Listing-Nr. 7
```

Achtung: Dieses Programm läuft nur, wenn vorher mit

```
RUN"LABEL 128"
```

die Routine »Label 128« geladen und aktiviert wurde.

INSTR – Strings untersuchen

Da eine Funktion, um die Position eines bestimmten Zeichens innerhalb einer Zeichenkette (String) zu ermitteln oder nur zu prüfen, ob dieses Zeichen überhaupt vorhanden ist, in Basic 2.0 schmerzlich vermißt wurde, haben sich sicherlich die meisten Programmierer ein eigenes Unterprogramm dafür angefertigt.
In Basic 7.0 steht nun eine eigene Funktion dafür zur Verfügung (wie in Simon's Basic mit PLACE), die natürlich wesentlich schneller arbeitet als ein in Basic geschriebenes Unterprogramm. Sie lautet nun

```
INSTR (a$,b$)
```

Dabei ist »b$« durch den String (oder die String-Variable) zu ersetzen, die in der Zeichenkette »a$« gesucht werden soll. Das Ergebnis der Funktion ist dann die Position des ersten Auftretens von b$ in a$; das Ergebnis 0 bedeutet, daß »b$« in »a$« nicht gefunden wurde.
Beispiel:

```
PRINT INSTR("MARKT & TECHNIK","TECHNIK")
```

meldet die Zahl 9, weil der Text »TECHNIK« in der Zeichenkette »MARKT & TECHNIK« an neunter Position steht.
Zusätzlich kann noch festgelegt werden, ab welcher Position im zu durchsuchenden String begonnen werden soll:

```
PRINT INSTR("MARKT & TECHNIK","TECHNIK",10)
```

sucht »TECHNIK« in »MARKT & TECHNIK« ab der 10. Position und wird dabei nicht fündig (Ergebnis: 0), da der gesuchte Text ab der 10. Stelle nicht mehr auftaucht.
In der Regel wird man einzelne Zeichen suchen:

```
PRINT INSTR("MARKT & TECHNIK","&")
```

liefert das Ergebnis 7.
Beschäftigen wir uns noch mit der Möglichkeit, daß INSTR den Wert 0 liefert. Zunächst eine Zusammenfassung der möglichen Ursachen:

- Die Position, ab der gesucht werden soll, ist größer als die Länge der zu durchsuchenden Zeichenkette. Beispiel:

  ```
  PRINT INSTR("XXX","X",10)
  ```

- Der zu durchsuchende String »a$« ist ein Leerstring. Beispiel:

  ```
  PRINT INSTR("","SUCHSTRING")
  ```

- Der Normalfall: Der Suchstring »b$« ist in »a$« nicht enthalten (oder zumindest nicht ab der Suchposition). Beispiele:

  ```
  PRINT INSTR("123456","A")
  PRINT INSTR("123456","1",3)
  ```

  (»1« ist zwar in »123456« enthalten, aber nicht ab der 3. Position)

Tips und Tricks zu INSTR

a) Häufigkeit eines Zeichens in einem String ermitteln

Folgendes Beispielprogramm fordert Sie zur Eingabe eines Strings auf und sagt dann, wie oft der Buchstabe »E« darin vorkommt:

```
10 INPUT "STRING";S$
20 P=1:REM  Bei Position 1 beginnen
30 Z=-1:REM  Zähler initialisieren (da Z in 50 erhöht wird, muß der Initialisierungs-
             wert "-1" sein, damit nach dem ersten Durchlauf von Zeile 50 die
             Variable Z den Wert 0 hat)
40 DO WHILE P<>0
50 :Z=Z+1:REM Zähler erhöhen
60 :I=INSTR(S$,"E",P):REM "E" in S$ suchen
70 :IF I<>0 THEN P=I+1:ELSE P=0
80 LOOP
90 PRINT Z;"MAL KOMMT 'E' VOR.":REM Zähler ausgeben
```

Dieses Programm kann leicht für eigene Zwecke abgeändert werden (beispielsweise zu einem Unterprogramm umfunktioniert werden).

b) INSTR in Verbindung mit IF, WHILE und UNTIL

In 3.3 wurde schon erwähnt, daß der IF-Befehl immer dann verzweigt, wenn ihm ein Ausdruck folgt, der einen anderen Wert als 0 (z.B. -1) hat. Zur Wiederholung mag folgendes Beispiel dienen:

```
10 INPUT "IF-WERT";IW:IF IW THEN PRINT "JA (WERT <> 0)":ELSE PRINT "NEIN
(WERT = 0)"
```

Nach dem gleichen Prinzip funktionert auch WHILE: Ist der Wert nach WHILE gleich -1, wird die DO-LOOP-Schleife fortgesetzt, ansonsten abgebrochen.
UNTIL funktioniert auf die gleiche Weise, allerdings in entgegengesetzter Richtung: Ist der Wert nach UNTIL ungleich 0 (Bedingung erfüllt), so wird die Schleife abgebrochen, bei Wert = 0 fortgesetzt.
Da INSTR den Wert 0 liefert, wenn die Suche erfolglos verlief, kann mit IF unkompliziert geprüft werden, ob der String 1 im String 2 vorkommt:

```
10 INPUT "STRING 1 (SUCHSTRING)";S1$:INPUT "STRING 2";S2$
20 IF INSTR(S2$,S1$) THEN PRINT "STRING 1 IST IN STRING 2 ENTHALTEN": ELSE
PRINT "STRING 1 IST IN STRING 2 NICHT ENTHALTEN"
```

Es ist also überflüssig, hinter »IF INSTR(S2$,S1$)« noch den Vergleich »<>0« anzugeben, da dies der IF-Befehl sozusagen automatisch erledigt.
Dies gilt auch für WHILE und UNTIL:

```
DO (bzw. LOOP) WHILE INSTR(....)
```

ist gleichbedeutend mit

```
DO (bzw. LOOP) WHILE INSTR(....)<>0
```

Entsprechend gilt:

```
DO (bzw. LOOP) UNTIL INSTR(....)
```

kann verwendet werden für

```
DO (bzw. LOOP) UNTIL INSTR(....)<>0
```

Das Weglassen von »<>0« ist sogar, sofern es erlaubt ist, die effektivere Lösung, da das Programm dadurch sogar geringfügig beschleunigt wird.
Unsere Hilfsroutine zur Ermittlung der Häufigkeit des Auftretens von »E« in einem bestimmten String kann deshalb noch optimiert werden:

```
10 INPUT "STRING";S$
20 P=1:REM Bei Position 1 beginnen
30 Z=-1:REM Zähler initialisieren
40 DO WHILE P:REM "WHILE P" statt "WHILE P<>0"
50 :Z=Z+1:REM Zähler erhöhen
60 :I=INSTR(S$,"E",P):REM "E" in S$ suchen
70 :IF I THEN P=I+1:ELSE P=0:REM "IF I" statt "IF I<>0"
80 LOOP
90 PRINT Z;"MAL KOMMT 'E' VOR.":REM Zähler ausgeben
```

Der Vollständigkeit halber sei noch eine UNTIL-Version von gleicher Wirkung vorgestellt:

```
10 INPUT "STRING";S$
20 P=1
30 Z=-1
40 DO UNTIL P=0:REM oder: "DO UNTIL (NOT P)"
50 :Z=Z+1
60 :I=INSTR(S$,"E",P)
70 :IF I THEN P=I+1:ELSE P=0:REM "IF I" statt "IF I<>0"
80 LOOP
90 PRINT Z;"MAL KOMMT 'E' VOR.":REM Zähler ausgeben
```

GO64 - Der Sprung in den C64-Modus

In den C64-Modus kommt man nicht nur, wenn man beim Einschalten oder einem Reset die <CBM>-Taste gedrückt hält, sondern auch durch Eingabe des Befehls

```
GO 64
```

Dieser Befehl wird in 5.1 ausführlich erläutert. An dieser Stelle nur soviel: Im Direktmodus erfolgt die Sicherheitsabfrage »ARE YOU SURE?«, die Sie von den Diskettenbefehlen HEA-

DER und SCRATCH kennen dürften (<Y>, bestätigt, eine andere Eingabe führt zurück in den Eingabe-Modus); innerhalb eines Programms wird der C64-Modus ohne Verzögerung angesprungen, allerdings das Programm nicht im C64-Modus fortgesetzt, sondern gelöscht!

BANK - Speicherbank anwählen

Aufgrund der Speicherstruktur des C128, die in 3.2 erklärt wird (siehe auch Anhang B des Handbuches), ist zusätzlich zur Angabe einer Adresse noch die Auswahl einer 64K-Speicherzusammensetzung erforderlich, um einen Speicherplatz eindeutig zu bestimmen. Für die Befehle PEEK, POKE, WAIT und SYS kann diese Speicherbank mit einem weiteren Basic-Befehl ausgewählt werden:

```
BANK n
```

stellt Speicherbank »n« (0-15) ein. Dabei sind in der Grundversion nur die Banks 0, 1, 14 und 15 von Basic aus sinnvoll einzusehen:

Bank 0: Dort wird das Basic-Programm gespeichert.
Bank 1: Speicherbank für Basic-Variablen
Bank 14: wie Bank 15, aber Zeichensatz-ROM bei 53248 verfügbar
Bank 15: Speicherbank für ROM (Basic-Interpreter, Betriebssystem)

Für Zugriffe auf die Adressen 0-1023 erübrigt sich die Bank-Einstellung, da dies der allen Banks gemeinsame Bereich von 1K Länge ist. Bank 0 und Bank 14/15 haben zudem den Bereich 1025-16383 gemeinsam, also 16K.
Die aktuelle über BANK eingestellte Speicherbank ermittelt die Funktion

```
PEEK(981)
```

PRINT PEEK(981) ergibt im Einschaltzustand 15.
Da die Adresse 981 im gemeinsamen Bereich 0-1023 liegt, ist ein Bank-Befehl nicht nötig.

STASH, SWAP und FETCH - Steuerung der RAM-Disk

Die Befehle

```
FETCH
STASH
SWAP
```

können nur ausgeführt werden, wenn eine RAM-Disk (Speichererweiterung) als Steckmodul angeschlossen ist. Da zum Zeitpunkt der Erstellung dieses Buches die Speichererweiterung (RAM-Disk) noch nicht erhältlich ist, muß ich Sie auf das C128-Handbuch verweisen, wo diese drei Befehle beschrieben werden:

FETCH Seite 4-61
STASH Seite 4-126
SWAP Seite 4-127

Unter Umständen werden diese Befehle auch im Handbuch zu Ihrer RAM-Disk besprochen; da mir dieses nicht vorliegt, kann ich Ihnen die Seitenzahlen folglich aber noch nicht nennen.

QUIT und OFF – Geheimnisvolle Befehle

Die Befehle QUIT und OFF werden im Handbuch nicht beschrieben, sind aber vorhanden. Die Eingabe von QUIT oder OFF führt zur Fehlermeldung

```
?UNIMPLEMENTED COMMAND ERROR
```

Diese wird ebenfalls im Handbuch nicht erwähnt. Offensichtlich sind diese Befehle erst für den Betrieb mit einer zukünftigen Erweiterung, vielleicht der RAM-Disk, vorgesehen. In der Grundversion des C128 ist jedoch nachweislich keine Routine zu diesen Befehlen vorhanden.

SLEEP – Der Pausenbefehl

Wie der Name »SLEEP« (engl. »schlafen«) schon vermuten läßt, kann mit diesem Befehl eine programmierte Unterbrechung erfolgen. Dazu muß die Dauer der Pause angegeben werden, die übrigens durch den FAST-Befehl nicht beeinflußt wird:

```
SLEEP n
```

»n« ist ein Wert zwischen 1 (1 Sekunde) und 65535 (65535 Sekunden, also ca. 18 Stunden!), der die Wartezeit in Sekunden angibt. Leider sind hier nur ganzzahlige Werte erlaubt.
Durch den SLEEP-Befehl werden Verzögerungsschleifen (»FOR I=1 TO 2000:NEXT I«) überflüssig; um aber Wartezeiten unter 1 Sekunde zu programmieren, kann SLEEP nicht eingesetzt werden.
Beispiele:

```
SLEEP 1
```

läßt den C128 ziemlich genau 1 Sekunde lang warten.

```
SLEEP 60
```

hält den Computer 1 Minute (60 Sekunden) lang an.
SLEEP kann übrigens mit <RUN/STOP> unterbrochen werden, wie jeder andere Basic-Befehl auch. Dies ist dann angebracht, wenn man sich mit der Wartezeit vertan hat.

RREG – Werte von Maschinenprogrammen an Basic übergeben

Während der SYS-Befehl aufgrund seiner neuen Syntax (siehe 3.2) zur Übergabe von Werten aus einem Basic- an ein Maschinenprogramm fähig ist, dient RREG der umgekehrten Kommunikation: RREG liest die 8510-Prozessorregister (Akkumulator, X- und Y-Register, Statusbyte) aus. Dieser Befehl ist also nur für Maschinenprogrammierer interessant; wer seinen C128 nicht in Maschinensprache programmiert, möge bitte bei der Erklärung der Befehle »FAST« und »SLOW« weiterlesen.

Zurück zu RREG. Mit

```
RREG A,X,Y,S
```

bekommen die Variablen A,X,Y und S folgende Werte zugewiesen:

A Zustand des Akkumulators bei Beendigung des letzten SYS-Befehls
X entsprechend für X-Register
Y entsprechend für Y-Register
S entsprechend für Statusregister

Statt A,X,Y und S können auch andere Variablennamen oder Variablentypen (Integer, Arrayelemente) eingesetzt werden. Soll ein bestimmtes Register nicht ausgelesen werden, muß auch an dessen Position kein Variablenname angegeben werden:

```
RREG,,Y,S
```

liest nur Y-Register und Statusregister aus (Akku und X werden durch die zwei Kommas übersprungen).

```
RREG A
```

liest nur den Akkumulator aus (Kommas am Ende der Anweisung können entfallen).
Die Syntax wird auch auf Seite 4-103 des C128-Handbuches aufgeführt.

FAST und SLOW - Die Gangschaltung des C128

Schon in Abschnitt 2.2.4 haben wir die Befehle FAST und SLOW, die die Arbeitsgeschwindigkeit des C128 regulieren, besprochen. Deshalb erübrigt sich eine erneute Erklärung.

XOR - Entweder-Oder als Funktion

Von Basic 2.0 kennen Sie die folgenden beiden Arten der logischen Verknüpfung, die bei IF-Befehlen häufig eingesetzt werden (auch bei WHILE und UNTIL in Basic 7.0):

NOT	NICHT-Verknüpfung
AND	UND-Verknüpfung
OR	ODER-Verknüpfung

NOT A	ist wahr, wenn A falsch ist.
A AND B	ist wahr, wenn A und B wahr sind.
A OR B	ist wahr, wenn A oder B oder beides wahr ist.

Dazu können Sie auch im C128-Handbuch in Kapitel 2.7.3 (Seiten 2-11 bis 2-13) nachlesen. Der Unterschied zwischen OR und XOR ist nun, daß A OR B auch dann wahr ist, wenn sowohl A als auch B zutreffen. A XOR B hingegen ist nur dann wahr, wenn A wahr und B falsch ist oder A falsch und B wahr ist. Es ergeben sich folgende Wahrheitstabellen (w=wahr, f=falsch) als Tabelle 3.4:

Tabelle 3.4: *Wahrheitstabellen von OR und XOR im Vergleich*

A	B	A OR B		A	B	A XOR B	
f	f	f		f	f	f	
w	f	w		w	f	w	
f	w	w		f	w	w	
w	w	w	<=>	w	w	f	(!)

Ein Beispiel hierfür in Worten:
Der C128 befindet sich entweder im FAST- oder im SLOW-Modus. Eine weitere Möglichkeit gibt es nicht, er kann aber auch nicht in beiden Modi gleichzeitig laufen.
Dies wäre ein typischer Fall von XOR:

C128 im FAST-Modus XOR C128 *im SLOW-Modus* ist wahr

Der Befehl XOR kann jedoch zum Leidwesen der Basic-Programmierer nicht wie AND und OR zwischen zwei Aussagen stehen:

```
IF A=5 XOR B=7 THEN ...
```

ist nicht zulässig (»?SYNTAX ERROR«).
Dafür kann aber folgende Hilfskonstruktion verwendet werden (auch in Basic 2.0), die auf der numerischen Auswertung von logischen Ausdrücken beruht:

```
IF (A=5)<>(B=7) THEN ...
```

Zusätzlich steht XOR als Basic-Funktion zur Verfügung, wenn zwei Zahlen bitweise zu verknüpfen sind:

```
XOR (255,154)
```

verknüpft die Zahlen 255 und 154 bitweise (Ergebnis: 101). Zugelassen sind Zahlen von 0 bis 65535. Im Gegensatz dazu arbeiten AND und OR in Verbindung mit Zahlen mit dem Bereich von -32767 bis +32767.
Ein A XOR B, das mit dem Zahlenbereich von NOT, AND und OR arbeitet, wird mit Hilfe dieser Befehle folgendermaßen simuliert:

```
((NOT A) AND B) OR ((NOT B) AND A)
```

Wer sich mit Bitklaubereien auskennt, kann XOR beispielsweise zur Codierung von Texten verwenden. Da im Rahmen dieses Buches auf das Thema »Logische Operatoren« nicht weiter eingegangen werden kann, sei Interessierten die Artikelserie »Logeleien«, die in der Zeitschrift 64'er in den Ausgaben 7/85-9/85 erschien. Dort wird die Anwendung von AND, OR, XOR und NOT ausführlich und anhand vieler Beispiele erläutert.
Eine knappe Erklärung, die die Kenntnis des Binärsystems teilweise voraussetzt, gibt das C128-Handbuch in Abschnitt 2.7.3.

POINTER - Wieder etwas für Maschinensprachler

Da auch der POINTER-Befehl hauptsächlich für Maschinensprache-Zwecke von Bedeutung ist (um die Adresse einer Variablen in Bank 1 zu übergeben) und somit nur einen Teil der Leser betrifft, soll dieser an dieser Stelle nicht weiter behandelt werden. Eine Beschreibung dieses für Profis nützlichen Kommandos steht auf Seite 4-86 des C128-Handbuches.

3.4.7. Übersicht über die neuen Befehle

Nachdem in 3.4.1-3.4.6 diejenigen Befehle, die für uns als Umsteiger vom C64 neu sind (die Basic-2.0-Befehle sind voll in Basic 7.0 integriert), besprochen wurden, soll Tabelle 3.5 einen kleinen Überblick darstellen. Dazu sind die Befehle, die gegenüber Basic 2.0 (C64/VC20) hinzugekommen sind, im folgenden alphabetisch aufgeführt. In Klammern steht hinter jedem Befehl das Kapitel dieses Buches, in dem der Befehl besprochen wird (3.4.1-3.4.6) und dahinter, durch ein Semikolon getrennt, die Seitenzahl im C128-Handbuch, damit Sie die dortige Befehlsübersicht schnell finden können.

Tabelle 3.5: *Basic-7.0-Zusatzbefehle im Überblick*

APPEND	(3.4.5;4-13)	SEQ-Datei zum Datenanfügen öffnen
AUTO	(3.4.1;4-14)	automatische Zeilennumerierung
BACKUP	(3.4.5;4-15)	Diskseite mit Doppelfloppy kopieren
BANK	(3.4.6;4-17)	Speicherbank (POKE usw.) wählen
BEGIN...BEND	(3.4.2;4-18)	hinter THEN/ELSE mehrere Zeilen als Block zusammenfassen
BLOAD	(3.4.5;4-19)	Maschinenprogramm laden
BOOT	(3.4.5;4-21) (auch 7.3)	Maschinenprogramm laden und starten oder BOOT-fähiges Programm laden
BOX	(3.4.3;4-22)	Rechteck zeichnen
BSAVE	(3.4.5;4-24)	Maschinenprogramm abspeichern
BUMP-Funktion	(3.4.3;4-26)	Auswertung einer Sprite-Kollision
CATALOG	(3.4.5;4-27)	Directory (Inhaltsverzeichnis) der Diskette anzeigen (identisch mit DIRECTORY)
CHAR	(3.4.3;4-28) (auch 3.4.5)	Text drucken, auch im Grafikmodus
CIRCLE	(3.4.3;4-30)	Kreis, Ellipse oder Vieleck zeichnen
COLLECT	(3.4.5;4-32)	Diskettenbefehl "VALIDATE"
COLLISION	(3.4.3;4-33)	dient Abfrage von Spritekollisionen
COLOR	(3.4.3;4-35) (auch 3.4.5)	setzt Farben für Text und Grafik

CONCAT	(3.4.5;4-36)	zwei Dateien verbinden
COPY	(3.4.5;4-37)	Diskettenbefehl "COPY"
DCLEAR	(3.4.5;4-38)	alle Floppy-Kanäle schließen
DCLOSE	(3.4.5;4-39)	Floppy-Kanal schließen
DEC-Funktion	(3.4.6;4-40)	Dezimalwert einer Hexadezimalzahl
DELETE	(3.4.1;4-41)	Zeilenbereich aus Programm löschen
DIRECTORY	(3.4.5;4-42)	Disk-Inhaltsverzeichnis, wie CATALOG
DLOAD	(3.4.5;4-44)	Programm von Diskette laden
DO...LOOP	(3.4.2;4-45)	Programmschleifen
DOPEN	(3.4.5;4-47)	Diskettendatei öffnen
DRAW	(3.4.3;4-49)	Punkt oder Linie zeichnen/löschen
DSAVE	(3.4.5;4-51)	Programm auf Diskette speichern
DS-Variable	(3.4.5;4-52)	liefert den Fehlerstatus der Floppy als Code (0=kein Fehler)
DS$-Variable	(3.4.5;4-52)	wie DS, aber als Meldung
DVERIFY	(3.4.5;4-53)	Programm auf Diskette verifizieren
EL-Variable	(3.4.6;4-55)	enthält Zeilennummer bei Fehler
ELSE	(3.4.2;4-73)	Alternative bei IF...THEN
ENVELOPE	(3.4.4;4-56)	Hüllkurve für Soundeffekte setzen
ER-Variable	(3.4.6;4-58)	enthält Fehlercode bei Fehler
ERR$-Funktion	(3.4.6;4-59)	liefert Fehlermeldung im Klartext
EXIT	(3.4.2;4-45)	dient zum Verlassen der DO-Schleife
FAST	(3.4.6;4-60)	schaltet auf höhere Geschwindigkeit
FETCH	(3.4.6;4-61)	holt Daten aus Speicherbank (nur mit RAM-Floppy)
FILTER	(3.4.4;4-62)	setzt Klangfilter für Sound
GETKEY	(3.4.5;4-64)	wie 10 GET A$:IF A$="" THEN 10
GO64	(3.4.6;4-65)	schaltet in C64-Modus
GRAPHIC	(3.4.3;4-66)	schaltet Grafikmodus ein
GSHAPE	(3.4.3;4-68)	kopiert Shape aus String in Grafik
HEADER	(3.4.5;4-69)	Diskettenbefehl "NEW" (formatieren)
HELP	(3.4.1;4-71)	zeigt Fehlerzeile nach Meldung an
HEX$-Funktion	(3.4.6;4-72)	wandelt Dezimalzahl in Hexa-String
INSTR-Funktion	(3.4.6;4-75)	ergibt Position des Teilstrings in einem anderen String
JOY-Funktion	(3.4.5;4-76)	fragt Joystick ab
KEY	(3.4.1;4-77) (auch 3.4.5)	belegt Funktionstaste mit zeigt Belegungen an
OCATE	(3.4.3;4-78)	setzt Pixel-Cursor für Grafik

MID$	(3.2 ; -)	ermöglicht jetzt auch Wertzuweisung
MONITOR	(3.4.1;4-79)	ruft Monitorprogramm auf
MOVSPR	(3.4.3;4-80)	Sprite bewegen oder positionieren
PAINT	(3.4.3;4-81)	füllt Grafikbereich aus
PEN-Funktion	(3.4.5;4-82)	Abfrage von Lichtgriffel (Lightpen)
PLAY	(3.4.4;4-84)	spielt Musikstring ab
POINTER-Funkt.	(3.4.6;4-86)	ergibt Adresse einer Variablen
POT-Funktion	(3.4.5;4-87)	fragt Drehregler (Paddle) ab
PRINT USING	(3.4.5;4-88)	formatierte Datenausgabe
PUDEF	(3.4.5;4-92)	definiert Zeichen für PRINT USING
RCLR-Funktion	(3.4.3;4-94)	liefert Farbcode zu Farbquelle
RDOT-Funktion	(3.4.3;4-95)	liefert Werte zum Pixel-Cursor
RECORD	(3.4.5;4-96)	positioniert Schreib-/Lesezeiger
RENAME	(3.4.5;4-97)	Diskettenbefehl "RENAME"
RENUMBER	(3.4.1;4-98)	Programm umnumerieren
RESUME	(3.4.6;4-101)	Ende von Fehlerbehandlungsroutine
RGR-Funktion	(3.4.3;4-102)	liefert den aktuellen Grafikmodus
RREG	(3.4.6;4-103)	holt Prozessorregister
RSPCOLOR-Funkt.	(3.4.3;4-104)	liefert Codes für Sprite-Farben
RSPPOS-Funktion	(3.4.3;4-105)	liefert Sprite-Position oder die aktuelle Sprite-Geschwindigkeit
RSPRITE-Funkt.	(3.4.3;4-106)	liefert Sprite-Attribute
RWINDOW-Funkt.	(3.4.5;4-109)	liefert Parameter des Windows
SCALE	(3.4.3;4-110)	Maßstabswahl bei Grafik
SCNCLR	(3.4.3;4-111) (auch 3.4.5)	löscht Text- oder Grafikbildschirm
SCRATCH	(3.4.5;4-112)	Diskettenbefehl "SCRATCH"
SLEEP	(3.4.6;4-114)	hält Programmausführung an
SLOW	(3.4.6;4-115)	schaltet von FAST (s.dort) zurück
SOUND	(3.4.4;4-116)	erzeugt Ton zu Frequenz und Dauer
SPRCOLOR	(3.4.3;4-118)	setzt Multicolor-Farben für Sprite
SPRDEF	(3.4.3;4-119)	Sprite-Editor aufrufen
SPRITE	(3.4.3;4-122)	Sprite-Attribute setzen
SPRSAV	(3.4.3;4-124)	Sprite in String speichern
SSHAPE	(3.4.3;4-125)	speichert Shape in Stringvariable
STASH	(3.4.6;4-126)	Daten in RAM-Floppy übertragen (nur mit RAM-Disk!)
SWAP	(3.4.6;4-127)	Daten zwischen Bank und Speicher austauschen (nur RAM-Disk!)
TEMPO	(3.4.4;4-128)	Abspieltempo für PLAY setzen
TRAP	(3.4.6;4-129)	eigene Fehlerbehandlung aktivieren

TROFF	(3.4.1;4-130)	Programmablaufverfolgung aus
TRON	(3.4.1;4-130)	Programmablaufverfolgung ein
UNTIL	(3.4.2;4-45)	Bedingung für DO/LOOP (DO UNTIL...)
VOL	(3.4.4;4-131)	Lautstärke für Sound setzen
WHILE	(3.4.2;4-45)	Bedingung für DO/LOOP (DO WHILE...)
WIDTH	(3.4.3;4-132)	Strichstärke für Grafik setzen
WINDOW	(3.4.5;4-133)	Bildschirmfenster (Window) setzen
XOR-Funktion	(3.4.6;4-134)	Entweder-Oder-Verknüpfung

Die Fehlermeldungen des Basic 7.0 sind in 3.4.6 bei der Erklärung der ER-Systemvariablen (siehe TRAP) als Tabelle 3.3 aufgeführt.
Die Fehlermeldungen der Floppy können Sie dem Handbuch zum Diskettenlaufwerk entnehmen.

3.5 Umschreiben von C64-Basicprogrammen

Da der volle Basic-2.0-Befehlsvorrat im Basic 7.0 des C128 integriert ist, können C64-Basicprogramme, die es ja wie Sand am Meer gibt, auch auch auf dem C128 ablaufen, wenn vorher die C64-spezifischen Programmteile entfernt bzw. durch C128-Äquivalente ersetzt wurden.
Wie man dies bewerkstelligt und so C64-Programme auf dem C128 laufen läßt, ohne den C64-Modus anzusteuern, soll in diesem Abschnitt 3.5 gezeigt werden. Es können aber selbstverständlich nicht alle Probleme, die bei der Anpassung auftreten, behandelt werden. Die wichtigsten Fragen dürften aber geklärt werden, da ich u.a. auf meine eigene Erfahrung beim Umschreiben von C64-Programmen auf den C128 bauen kann.
Eins gleich vorweg: Die Vermutung, daß »reine« Basicprogramme, d.h. Programme, die die Befehle PEEK, POKE, WAIT und SYS nicht verwenden, im 40-Zeichen-Modus immer ohne Änderung ablaufen, ist leider ein Trugschluß, da manche Programme Schwierigkeiten machen; allerdings geht das Umschreiben bei solchen Programmen, wenn es nötig ist, viel leichter vonstatten, da nur die Bildschirmausgabe angepaßt werden muß, was in 3.5.2 erläutert wird.
Voraussetzung für diesen Abschnitt ist, daß Sie die Befehle des Basic 7.0 kennen; andernfalls können Sie sich mit Hilfe von Kapitel 3.4 das erforderliche Wissen aneignen.
Es sei noch erwähnt, daß ein Basic-2.0-Programm mit dem DLOAD-Befehl in den C128 geladen wird und sofort editierfähig ist (LIST, RENUMBER, DELETE usw.). Sie brauchen also kein Konvertierungsprogramm, da sich Disketten- bzw. Kassettenformat des C128 nicht vom C64 unterscheiden.

3.5.1 Die »DATA-Wüsten« - typisch C64!

Ein großes Problem stellen die endlos langen Programmteile aus DATA-Zeilen, etwas abschätzig »DATA-Wüsten« genannt, dar, die in C64-Programmen im Übermaß zu finden sind. Dabei müssen Sie zuerst ermitteln, welche Funktion die DATA-Werte haben, was bei eigenen Programmen kein Problem ist, bei fremden Programmen allerdings in mühsame Tüftelei ausarten kann. Es gibt viele verschiedene Funktionen von DATAs, die sich in der Regel nur an den dazugehörigen READ-Befehlen erkennen lassen. Ist die READ-Schleife gefunden (bei vielen DATA-Werten sind Schleifen zum Einlesen erforderlich), läßt sich die Funktion der DATA-Werte ermitteln. Zunächst klären wir, wie man anhand der READ-Schleife die Bedeutung erkennt, dann wird das Umschreiben selbst behandelt.

Maschinenprogramm

Erkennungsmerkmale:

Die DATA-Werte werden über Schleifen eingelesen, die zuerst den Wert aus der DATA-Zeile holen (READ) und diesen dann in den Speicher schreiben (POKE).
Beispiele von READ-Schleifen zu Maschinenprogrammen:

```
FOR I=49152 TO 51143:READ A:POKE I,A:NEXT
FOR I=1 TO 3248:READ A:POKE 49152+I,A:NEXT
```

Zwischen FOR und NEXT kann auch eine Prüfsummenberechnung stehen:

```
FOR I=38528 TO 38900:READ A:POKE I,A:S=S+A:NEXT
```

Noch ein Tip: Wenn in DATA-Zeilen gehäuft die Werte 169 und 141 (nicht nacheinander, sondern verteilt) stehen, handelt es sich ganz sicher um ein Maschinenprogramm.
Da aber READ-POKE-Schleifen außer Maschinenprogramme auch anderes einlesen können, wie wir noch sehen werden, ist dies allein kein unumstürzliches Merkmal. Das eindeutige Kennzeichen eines Unterprogramms in Maschinensprache ist, daß es von Basic aus über SYS aufgerufen wird. Steht in einem Basic-Programm kein SYS-Befehl, ist auch kein Unterprogramm in Maschinensprache vorhanden.
Andernfalls entscheidet die Zahl, die nach dem SYS-Befehl steht:
SYS-Parameter im Bereich 40960-49152 oder 57344-65535:
kein Maschinenprogramm, das speziell vom Basic-Programm über DATA eingelesen wurde, sondern ein ohnehin im C64 integriertes Maschinenprogramm aus Betriebssystem oder Basic-Interpreter.
SYS-Parameter im Bereich 512-40959 oder 49152-53247:
Maschinenprogramm, das vom Basic-Programm erzeugt wurde.
Steht ein SYS-Befehl unmittelbar nach der READ-Schleife, so ist dies ein sehr sicheres Kennzeichen, daß sich in den betreffenden DATA-Zeilen die Werte für ein Maschinenprogramm befinden. Beispiel:

```
FOR C=828 TO 900:READ X:POKE C,X:NEXT C:SYS 828
```

Umschreiben von Maschinenroutinen in DATAs:

Wer der Maschinensprache des C64/128 nicht mächtig ist, muß in diesem Fall passen: Maschinenroutinen sind im Grunde nie übertragbar und müssen immer umgeschrieben werden, was nur mit Maschinensprachekenntnissen möglich ist. Die einzige Hoffnung ist noch, die Maschinenroutinen durch gegenüber dem C64 hinzugekommene Basic-7.0-Anweisungen zu ersetzen, was im Bereich der Grafikprogrammierung in sehr vielen Fällen möglich ist. Dazu muß allerdings die genaue Funktionsweise und Syntax solcher SYS-Erweiterungen bekannt sein, was oft der Fall ist.
Assemblerfreaks können in 4.2 nachlesen, wie Maschinenprogramme angepaßt werden.

Sprite-Daten

Erkennungsmerkmale:

- Sprite-Daten werden meist im Bereich 673-767/820-1019 abgelegt
- Sprites werden über POKEs im Bereich 53248-53294 definiert
- Sprite-Daten werden wie Maschinenprogramme eingelesen
- Gehäuftes Auftreten der Werte 0 und auch 255 in den DATA-Zeilen

Anpassung der Einleseschleifen für Sprite-DATAs:

Ein Verändern der Sprite-Datas selbst ist nicht nötig, deshalb müssen nur die Einleseschleifen und Spritesteuerungsbefehle umgeschrieben werden.
Wenn man die Sprites durch Verändern der READ-POKE-Schleifen in den Bereich 3584-4095 verlegt, hat man einen geeigneten Speicherbereich gefunden, wo ein Überschreiben der Sprites unmöglich ist.

Dann hat man folgende zwei Möglichkeiten:
- Man programmiert die Sprites weiterhin über POKEs (s. 3.5.4); dann müssen dem VIC auch die Adressen mitgeteilt werden.
- Man verwendet zur Sprite-Steuerung die komfortablen Befehle des Basic 7.0. Dann haben die Sprites in folgenden Bereichen zu liegen (Tabelle 3.6):

Tabelle 3.6: *Adressen der Sprites im Speicher*

Sprite 1: 3584-3647	(hexadezimal: $0E00-$0E3F)
Sprite 2: 3648-3711	(hexadezimal: $0E40-$0E7F)
Sprite 3: 3712-3775	(hexadezimal: $0E80-$0EBF)
Sprite 4: 3776-3839	(hexadezimal: $0EC0-$0EFF)
Sprite 5: 3840-3903	(hexadezimal: $0F00-$0F3F)
Sprite 6: 3904-3967	(hexadezimal: $0F40-$0F7F)
Sprite 7: 3968-4031	(hexadezimal: $0F80-$0FBF)
Sprite 8: 4032-4095	(hexadezimal: $0FC0-$0FFF)

Wie man die PEEK- und POKE-Befehle zur Sprite-Steuerung in die leistungsfähigen Befehle des Basic 7.0 umwandelt, wird in 3.5.4 zusammen mit den anderen PEEKs und POKEs behandelt.

Veränderter Zeichensatz

Erkennungsmerkmale:

- READ-POKE-Schleife wie bei Maschinenprogrammen und Sprites
- Adressen meist im Bereich 12288-16383
- Gehäuftes Auftreten der Werte 0 und 255 in den DATAs
- Zeichensätze werden über POKEs in die Adresse 53272 und manchmal zusätzlich über POKEs in 56576 aktiviert

Anpassung:

Wie bei den Sprites, ändert sich auch hier an den DATA-Werten selbst nichts. Aber an den POKEs, die den Zeichensatz einlesen und aktivieren, muß einiges an Veränderungen vorgenommen werden, was in der Regel nur denen möglich ist, die sich mit der Grafikprogrammierung entsprechend auskennen. Deshalb sind die folgenden Hinweise auch speziell an Grafikerfahrene gerichtet (andere werden sich bei der Anpassung einer Zeichensatzmanipulation an den C128 aufgrund dessen komplizierter Speicherstruktur die Zähne ausbeißen):

- Der Zeichensatz sollte bei 12288 beginnen oder woanders im Bereich 7168-16383 liegen. Dann kann er über folgende Befehle gegen Überschreiben durch Basic geschützt werden:

  ```
  GRAPHIC 1:GRAPHIC 0
  ```

  Allerdings darf dann die Grafik nicht benutzt werden, da sich die Speicherbereiche für Grafik und Zeichensatz überschneiden.

- POKEs in VIC-Register sind nur schwer möglich, was auch das für den Zeichensatz wichtige Register in 53272 betrifft. Näheres in 3.5.4.
- Der Zeichensatz muß immer in Bank 0 liegen.
- Ein veränderter Zeichensatz im 80-Zeichen-Modus hat zwar die gleichen Daten, wird aber anders aktiviert. Wie man dem VDC einen anderen Zeichensatz beibringt, wird in 3.8 angedeutet.
- Um einen Zeichensatz, der frühestens bei 16384 beginnt, abzusichern, erhöht man den Basic-Start. Diese POKEs sind

  ```
  POKE 45,...     statt:     POKE 43,...
  POKE 46,...     statt:     POKE 44,...
  ```

Daten für Basic-Zwecke

Erkennungsmerkmal:

- Daten werden nicht über POKE im Speicher abgelegt, sondern in Variablen aufgenommen oder von Basic-Befehlen (außer POKE) verarbeitet. Beispiele:

```
FOR I= 1 TO 1000:READ A$:PRINT A$;:NEXT
FOR C= 5 TO 150:READ D%(C):NEXT
FOR F= 0 TO 200:READ A,B,C:....:NEXT
```

Maßnahmen zur Anpassung an den C128:

In der Regel ist eine Anpassung an Basic 7.0 nicht erforderlich. Ausnahmen sind höchst unwahrscheinlich.

Sonstige Möglichkeiten

Durch die 4 genannten DATA-Funktionen sind 99%aller Fälle abgedeckt. Weitere Möglichkeiten sind nicht von allgemeinem Interesse und werden hier aus diesem Grund nicht behandelt. Bei eigenen Programmen dürften Sie aber auch ohne Erklärungen in der Lage sein, Spezialfälle zu meistern.

3.5.2 Bildschirmausgabe

Soll ein C64-Programm im 80-Zeichen-Modus laufen, ist es ratsam, die Bildschirmausgabe neu zu programmieren, damit die doppelte Zeichenbreite auch wirklich genutzt wird (z.B. für zusätzliche Informationen am Bildschirm, für die am 40-Zeichen-Bildschirm nicht ausreichend Platz vorhanden wäre), oder eine Lösung wie in 3.3 (Bildschirmausgabe in beiden Modi) wäre denkbar. POKEs in den Bildschirmspeicher sind dann im 80-Zeichen-Modus auch möglich, wie es in 3.8 gezeigt wird.
Bei Verwendung des 40-Zeichen-Bildschirms lohnt dies natürlich nicht, da es nur eine Schwierigkeit beim Umschreiben gibt: die »40-Zeichen-Überschreitung«. Damit ist gemeint, daß mehr als 40 Zeichen hintereinander (auch durch ein Semikolon getrennt) ausgegeben werden. In diesem Punkt reagieren C64 und C128 anders. Am besten sehen wir uns dies anhand eines kurzen Programms an, das hier den Unterschied zwischen C64 und C128 eindeutig aufweist.
Beschreibung: Demoprogramm für die 40-Zeichen-Überschreitung
Filename: »40-z.-ueberschr.«

```
100 REM *** DEMONSTRATIONSPROGRAMM ***
110 REM *** ZUR "40-ZEICHEN-UEBER- ***
120 REM *** SCHREITUNG" BEIM C 128 ***
130 :
200 REM 40 MAL "*" DRUCKEN:
210 :
220 FOR I = 1 TO 40
230 :PRINT "*";
240 NEXT I
250 :
300 REM CARRIAGE RETURN DRUCKEN:
310 :
320 PRINT :REM "PRINT"="PRINT CHR$(13);"
330 :
400 REM 20 MAL "*" DRUCKEN
410 :
420 FOR I = 1 TO 20
430 :PRINT "*";
440 NEXT I
450 :
```

Dieses Programm gibt zuerst 40 Sterne aus (Zeile 220), druckt dann ein RETURN (Wagenrücklauf) in Zeile 320 und schließlich 20 weitere Sterne. Wenn man dieses Programm, das nur aus Basic-2.0-Befehlen besteht, im C64-Modus laufen läßt, erhält man folgende Ausgabe:

```
****************************************
<1 Leerzeile>
********************
```

Dies liegt daran, daß das C64-Betriebssystem bei Erreichen der letzten Spalte – also nach 40 Zeichen – automatisch an den Anfang der nächsten Zeile springt; wenn dann noch, wie im Beispielprogramm, ein Zeilenvorschub ausgelöst wird, hat dies eine Leerzeile zur Folge.
Anders beim C128. Dort druckt das Programm folgendes aus:

```
****************************************
********************
```

Offensichtlich ist also ein Unterschied bei der Bildschirmausgabe zwischen C64 und C128 vorhanden. Dieser hat zur Folge, daß C64-Programme wie unser Beispielprogramm auf dem C128 an bestimmten Stellen eine Leerzeile nicht drucken, die auf dem C64 allerdings erscheint. Das Beispielprogramm hat dies gezeigt.
Wenn jetzt Leerzeilen wesentliche Bestandteile einer Bildschirmausgabe sind (z.B. weil die darauffolgenden Bildschirmausgaben sonst eine Zeile zu weit oben erfolgen würden), kann es sehr störend sein, wenn auf dem C128 eine Leerzeile fehlt. Als Beispiel sei nur der Fall genannt, daß eine Hintergrundgrafik für ein Spiel mit PRINT programmiert wird, wo das Fehlen einer Leerzeile den Bildschirmaufbau erheblich stört, ja sogar zu fehlerhaftem Verhalten des Programms führt.

Was kann man dagegen unternehmen?

Nun, die Lösung ist recht einfach:
Man sieht sich das Programm im C64-Modus an und merkt sich, wo Leerzeilen nötig sind. Dann fügt man im C128-Modus an den entsprechenden Stellen den Befehl »PRINT« ein. An unserem Beispielprogramm ist dies in Zeile 320 nötig:

```
320 PRINT:PRINT        statt:        320 PRINT
```

Nach dieser Änderung läuft das Programm auf dem C128 genauso ab, wie die Vorversion auf dem C64. Eine Lösung, die auf beiden Computern das gleiche bewirkt, ist aber nicht möglich.

3.5.3 Funktionstasten

Die komfortable Funktionstastenbelegung des C128 ist zwar als Eingabeerleichterung sehr hilfreich und kann auch in Programmen sinnvoll genutzt werden (Beispiele in 3.4.5), aber eine Abfrage der Funktionstasten mit Hilfe von CHR$-Codes ist nicht mehr möglich.
Schon in 2.1.1 wurde ein Verfahren vorgestellt, wie man die Funktionstastenbelegung auf die Codes 133-140 rückgängig macht:

```
KEY 1,CHR$(133):KEY 2,CHR$(137):KEY 3,CHR$(134):KEY 4,CHR$(138):
KEY 5,CHR$(135):KEY 6,CHR$(139):KEY 7,CHR$(136):KEY 8,CHR$(140)
```

Wenn man diese Befehle an den Anfang eines Programms setzt, damit die C64-Funktionstastenabfrage, die im C128-Handbuch auf Seite G-2 kurz wiederholt wird, möglich ist, erübrigt sich weiteres Umschreiben der eigentlichen Funktionstastenabfrage. Allerdings muß man auf die Funktionstastenbelegung mit GRAPHIC, LIST, DIRECTORY, RUN usw. verzichten, was den Eingabekomfort senkt.
Nach den genannten KEY-Befehlen gelten wieder die C64-üblichen Codes, die hier als Tabelle 3.7 vorgestellt werden.

Tabelle 3.7: *Werte der Funktionstasten nach Reduzierung auf CHR-Codes*

F1 = CHR$(133)
F2 = CHR$(137)
F3 = CHR$(134)
F4 = CHR$(138)
F5 = CHR$(135)
F6 = CHR$(139)
F7 = CHR$(136)
F8 = CHR$(140)

Nach CHR$-Codes geordnet:

133 = F1
134 = F3
135 = F5
136 = F7
137 = F2
138 = F4
139 = F6
140 = F7

3.5.4 PEEKs & POKEs anpassen

Die Befehle PEEK und POKE haben zwar beim C128 die gleiche Syntax wie beim C64, aber die Wirkung unterscheidet sich ganz erheblich. Da anspruchsvolle C64-Basic-Programme aufgrund der Mängel des Basic 2.0 ohne PEEK und POKE nicht auskommen (schon zum Ändern der Bildschirmfarben ist POKE erforderlich) und PEEK/POKE-Anweisungen meist nur für einen bestimmten Computer Gültigkeit haben, ist das Umschreiben von diesen wohl das größte Problem bei der Anpassung von C64-Programmen an den C128. Die PEEKs und POKEs lassen sich in folgende drei Gruppen gliedern:

- PEEKs und POKEs für Grafik
- PEEKs und POKEs für Sound
- PEEKs und POKEs für Basic

Unter c) fallen beispielsweise PEEK/POKE-Befehle zum Heraufsetzen der Anfangsadresse des Basic-Programms im Speicher; die Bedeutung von a) und b) geht aus obiger Beschreibung eindeutig hervor.
Im folgenden werden alle drei Gruppen getrennt behandelt, da aufgrund der Einteilung keine Bereichsüberschneidungen auftreten. Außerdem fällt dadurch ein späteres Nachschlagen leichter.

PEEKs und POKEs für Grafik

Hier ergibt sich das Problem, daß ein POKEn in die Register des VIC nicht möglich ist (der Ausgangszustand wird augenblicklich wiederhergestellt).

- *PEEKs und POKEs in den Bildschirmspeicher*

Der Bildschirmspeicher des 40-Zeichen-Modus liegt wie beim C64 in den Adressen 1024-2023. Damit diese zugänglich sind, muß Bank 0 oder, was aus verschiedenen anderen Gründen viel besser ist, Bank 15 eingeschaltet sein.
Da sich die Bildschirmcodes nicht geändert haben, solange man den amerikanischen Zeichensatz (ASCII-Zeichensatz) verwendet, und die Farbcodes für PEEK/POKE gleich geblieben sind (auch wenn sie sich von den COLOR-Farbcodes dadurch unterscheiden, daß sie jeweils um 1 niedriger sind), ergeben sich dann keine Probleme. Das POKEn des Farbcodes in den Farbspeicher (55296-56295), das bei älteren Versionen des C64 noch erforderlich war, ist beim C128 überflüssig. Soll allerdings die Schriftfarbe gewechselt werden, muß selbstverständlich auch der Farbcode gesetzt werden. Dies ist allerdings nur möglich, wenn Bank 15 eingestellt ist; deshalb sollte man zum POKEn in den Bildschirmspeicher auf jeden Fall diese Speicherkonfiguration anwählen, damit keine Probleme auftreten.
Die Manipulation des 80-Zeichen-Bildschirmspeichers ist etwas schwieriger zu programmieren als beim 40-Zeichen-Bildschirm; dennoch wird dies in 3.7.6 und 3.8 vorgestellt.

```
PRINT PEEK(2619)*4
```

ermittelt (Ergebnis: 1024).
Beim C64 müßte man »648« statt »2619« einsetzen; allerdings kann die Adresse 2619 des C128 kaum manipuliert werden, da der C128 nicht in der Lage ist, Eingaben aus einem anderen Bildschirmspeicher zu holen. Deshalb kann 2619 im Prinzip nur über PEEK ausgelesen werden, POKE bleibt relativ wirkungslos.

- *PEEKs und POKEs für Bildschirmfarben*

Solange Bank 15 eingeschaltet ist, können die VIC-Register über PEK und POKE beeinflußt werden. Da sich deren Programmierung in diesem Fall nicht geändert hat, können die Anweisungen

```
POKE 53280,Farbcode 0-15
```

und

```
POKE 53281,Farbcode 0-15
```

zum Einstellen der Rahmen- bzw. Hintergrundfarbe des 40-Zeichen-Bildschirms weiterverwendet werden. Dies sind aber so ziemlich die einzigen VIC-Register, die unmittelbar über POKE manipuliert werden können.
Aus Gründen der größeren Übersichtlichkeit sollte man jedoch den COLOR-Befehl verwenden:

```
COLOR 4,Farbcode 1-16        statt:        POKE 53280,Farbcode 0-15
COLOR 0,Farbcode 1-16        statt:        POKE 53281,Farbcode 0-15
```

Die Farbcodes des COLOR-Befehls sind um 1 höher als die entsprechenden Farbcodes bei Einsatz des POKE-Befehls. »COLOR 4,1« entspricht also »POKE 53280,1«.
Dies gilt auch für den auf dem C128 nicht verwendbaren POKE-Befehl zum Ändern der Textfarbe, der unbedingt durch COLOR zu ersetzen ist und dann sowohl im 40- als auch im 80-Zeichen-Modus arbeitet:

```
COLOR 5,Farbcode 1-16        statt:        POKE 646,Farbcode 0-15
```

Anweisungen der Form »PRINT CHR$(x)« zum Ändern der Textfarbe müssen nicht angepaßt werden. Beispiel: PRINT CHR$(5) stellt sowohl auf C64 als auch auf C128 (unabhängig von 40- oder 80-Zeichen-Modus) die weiße Textfarbe ein.
Die 80-Zeichen-Bildschirmfarben können durch »POKE 53280,...« bzw. »POKE 53281,...« nicht beeinflußt werden. Da beim 80-Zeichen-Bildschirm im Gegensatz zur 40-Zeichen-Darstellung nicht zwischen Rahmen-und Hintergrundfarbe unterschieden werden kann, genügt ein einziger COLOR-Befehl:

```
COLOR 6,Farbcode 1-16
```

Die Farbcodes im 80-Zeichen-Modus sind die COLOR-üblichen Codes.
Soll ein Programm in beiden Modi (40 und 80 Zeichen pro Zeile) lauffähig sein, müssen die Bildschirmfarben entsprechend für beides gesetzt werden. Beispiel:

```
COLOR 0,1:COLOR 4,1:COLOR 6,1
```

schaltet beide Bildschirme auf schwarz.
»COLOR 5,...« (Ändern der Textfarbe) ist aber Modus-unabhängig.
Die PEEK-Befehle in 53280, 53281 und 646 ersetzt man durch RCLR:

RCLR(0) entspricht PEEK(53281)
RCLR(4) entspricht PEEK(53280)
RCLR(5) entspricht PEEK(646)
RCLR(6) liefert den Farbcode des 80-Zeichen-Hintergrunds

Bei der RCLR-Funktion ist auf die Farbcodes zu achten; diese sind, wie beim Gegenstück COLOR, um 1 höher als die POKE-Codes.

- *PEEKs und POKEs für hochauflösende Grafik*

Zur Programmierung von hochauflösender Grafik über PEEK und POKE, wie dies beim C64 nötig ist, wenn man keine Basic-Erweiterung zur Verfügung hat, ist nur zu sagen, daß PEEKs und POKEs durch Basic-7.0-Befehle von gleicher Wirkung ersetzt werden sollen, da dies Verarbeitungsgeschwindigkeit und Übersichtlichkeit erhöht. Am besten ersetzt man PEEK-POKE-Unterprogramme durch Basic-7.0-Anweisungen und entfernt dann die Unterroutinen. Unverbesserlichen sei gesagt, daß der GRAPHIC-1-Modus des C128 den Farbspeicher für die hochauflösende Grafik nicht in den Bildschirmspeicher (1024-2023) legt, wie dies beim C64 der Fall ist, sondern daß die Farbinformationen im Bereich 7168-8191 liegen. Die Farbcodes für POKE haben sich nicht geändert, allerdings sind die entsprechenden Werte, die beim COLOR-Befehl eingesetzt werden, jeweils um 1 höher. Beispiel: schwarz bei PEEK/POKE: 0, bei COLOR: 1
Das gilt übrigens auch für den Farbspeicher des Textbildschirms (55296-56295, hexadezimal $D800-$DBFF).
Allerdings ist das Einschalten nicht mehr möglich, da der VIC an den entsprechenden Stellen nicht mehr direkt über PEEK und POKE beeinflußt werden kann. Deshalb muß man entweder auf den GRAPHIC-Befehl ausweichen oder die indirekte VIC-Register-Manipulation, die bei »PEEKs und POKEs zum Zeichensatz« gleich beschrieben wird, verwenden.

- *PEEKs und POKEs für Sprites*

Die Sprite-Register werden von Basic 7.0 so intensiv beeinflußt (jede 1/60 Sekunde erfolgt eine Neubelegung), daß ein Verändern über POKE nicht möglich ist. Dafür kann man aber komfortable Befehle einsetzen. So müssen beispielsweise die Multicolorfarben statt über

```
POKE 53248+37,Farbcode 1:POKE 53248+38,Farbcode 2
```

über folgenden Befehl eingestellt werden:

```
SPRCOLOR Farbcode 1, Farbcode 2
```

Dabei sind die unterschiedlichen Farbcodes von POKE und Basic-7.0-Befehlen zu beachten. Die Positionierung kann nicht mehr über POKEs in die Register 0-16 erfolgen, da MOVSPR diese Aufgabe hat. Um Sprite 1 an die Position x=200/y=200 zu bringen, schreibt man statt

```
POKE 53248+0,200:POKE 53248+1,200
```

folgendes:

```
MOVSPR 1,200,200
```

Die Bedeutung der einzelnen VIC-Register finden Sie in einer Tabelle im C128-Handbuch, Seiten E-2 und E-3.
Die Sprite-Befehle des Basic 7.0 werden in diesem Buch in Kapitel 3.4.3 besprochen; wo man die Sprite-Daten im C128-Speicher abzulegen hat, wird in 3.5.1 beschrieben. Mit Hilfe dieser Quellen dürfte es für Sie kein Problem mehr sein, Befehle zur Sprite-Steuerung umzuschreiben.

- *PEEKs und POKEs für einen veränderten Zeichensatz*

Als bequeme und schnelle Möglichkeit, abwechslungsreiche Grafiken darzustellen, sind die veränderten Zeichensätze ein beliebtes Hilfsmittel. Eine Anpassung an den C128 ist, wie in 3.5.1 schon gesagt wurde, relativ problematisch, was auch damit zusammenhängt, daß Basic 7.0 hierfür keine Befehle kennt, die die PEEK- und POKE-Befehle ersetzen könnten, wie dies etwa bei den Sprites möglich ist.
In 3.5.1 wurde schon ein Teil der erforderlichen Anpassungen genannt; offen blieb die Frage, wie man dem C128 die Lage des Zeichensatzes im Speicher mitteilt. Dabei ergibt sich nämlich das Problem, das die dafür so wichtige Adresse 53272 (VIC-Register 18) nicht mehr direkt manipuliert werden kann.
Allerdings kann dafür im Textmodus die Adresse 2604 (z.B. zum Verändern der Lage des Zeichensatzes), im Grafikmodus 2605 (z.B. zum Verändern der Lage der Bitmap) verwendet werden; diese Adressen sind ein gleichwertiger Ersatz für 53272, da sie direkt Einfluß auf das VIC-Register 24 (Adresse 53248+24=53272) nehmen, was über »POKE 53272,...« nicht mehr möglich ist.
Wenn für bestimmte Zwecke wie das Verändern der Lage des Zeichensatzes die Adresse 56576 (ein CIA-Register) mit PEEK und POKE geändert wird, ist keine Hilfslösung erforderlich, da Zugriffe auf die CIA-Register nicht angepaßt werden müssen.

- *Die zusätzlichen VIC-Register 47/48*

Da der VIC gegenüber dem C64 zwei neue Register bekommen hat (Register 47 und 48), kann ein POKEn in die Adressen 53295 oder 53296, was beim C64 keine Folgen hatte, unerwünschte Wirkung haben. Im Fall von 53295 (Register 47) ist dies zwar kaum möglich, aber durch unkontrolliertes Schreiben von Werten in 53296 kann ein Umschalten zwischen FAST- und SLOW-Modus erfolgen.

PEEKs und POKEs für Sound

Im Gegensatz zur Grafik, können sämtliche PEEK- und POKE-Befehle in den SID (Basisadresse: 54272, hexadezimal $D400) weiterverwendet werden. Voraussetzung ist, daß Bank 15 eingestellt ist.
Beim Umschreiben von C64-Programmen lohnt ein Einsatz der Basic-7.0-Befehle deshalb kaum, weil das Programm auch mit PEEK und POKE lauffähig ist. Für einzelne Soundeffekte kann aber der SOUND-Befehl geeignet sein, um das Programm zu verkürzen und/oder übersichtlicher zu gestalten, wie in 3.4.4 gezeigt wird.

PEEKs und POKEs für Basic

Damit werden fast alle noch übriggebliebenen Adressen behandelt, die durch PEEK/POKE angesprochen werden könnten.
Wir arbeiten uns dabei von niedrigen Adreßwerten zu höheren vor, da dies ein späteres Nachschlagen erleichtert.
Besprochen werden nur solche gängigen POKEs, die zu den typischen Tips und Tricks zählen

und sinnvolle Wirkungen haben (Absturz-POKEs oder andere, die die Funktionsweise beeinträchtigen, sind also belanglos).

- *Adresse 19: Fragezeichen bei INPUT unterdrücken*

Mit

```
POKE19,1:INPUT A$
```

kann man in C64-Programmen das Fragezeichen, das bei INPUT sonst automatisch ausgegeben wird, unterdrücken. Beim C128 ist dies mPOKE21,1:INPUT A$ möglich, allerdings sollte man sowohl bei C64 als auch beim C128 auf den POKE verzichten und

```
OPEN 1,0:INPUT!1,A$:CLOSE 1
```

verwenden, was denselben Effekt hat.

- *Adressen 43/44: Startadresse für Basic*

Beim C128 wird diese Funktion von 45/46 ausgeübt. Zu beachten ist, daß der Basic-Start des C64 bei 2049 liegt; beim C128 ist er 7169, sobald aber die hochauflösende Grafik verwendet wird, beginnt der Basic-Speicher in Bank 0 bei 16385.
Statt

```
POKE 43,...:POKE 44,...
```

heißt es beim C128 also

```
POKE 45,...:POKE 46,...
```

Allerdings muß, wie gesagt, in der Regel auch der POKE-Wert geändert werden.

- *Adressen 45/46: Startadresse für Variablen*

Während beim C64 die Adressen 45/46 aufgrund der Struktur des Basic-Speichers mit der Startadresse der Variablen gleichzeitig das Programmende bestimmten, ist dies beim C128 anders:

```
Adressen   47/  48: Startadresse für Variablen in Bank 1 (1024)
Adressen 4624/4625: Endadresse des Programms in Bank 0
```

Ein künstliches Heraufsetzen von 4624/4625, um das Nachladen eines Programms zu ermöglichen (Overlay-Technik), ist beim C128 nicht nötig, da die Variablen in Bank 1 liegen und somit das Programm in Bank 0 geschützt ist - unabhängig von 4624/4625.
Sie können also innerhalb eines Programms ohne besondere Vorkehrung einen weiteren Programmteil nachladen.

- *Adresse 22: Zeiger für temporären Stringstapel*

Zugriffe auf die Adresse 22 müssen beim C128 durch Operationen mit Adresse 24 ersetzt werden. Dies gilt auch für den Befehl

```
POKE 22,35                  (beim C128: POKE 24,35)
```

Dieser bewirkt, daß beim Listen eines Basic-Programms die Ausgabe der Zeilennummern unterdrückt wird. Beim Assembler TOP-ASS werden nach diesem POKE bei .E (nicht bei .LIST) die Zeilennummern durch ein Doppelkreuz (!) ersetzt.

– *Adresse 157: Direkt- oder Programm-Modus simulieren*

Diese Adresse ist gleichgeblieben. Sie dient bei der Ausgabe von Systemmeldungen als Informationsquelle, ob sich der Computer im Direktmodus oder im RUN-Modus befindet:

```
POKE 157,128        Direktmodus vortäuschen
POKE 157,0          Programm-Modus vortäuschen
```

Nach »POKE 157,128« werden alle Meldungen wie »SEARCHING« usw. ausgegeben und bei bestimmten Befehlen (HEADER, SCRATCH, GO64) erfolgt die Sicherheitsabfrage »ARE YOU SURE?«, da sich der C128 dadurch im Programm-Modus wähnt; »POKE 157,0« bewirkt das genaue Gegenteil. Allerdings sollte man bei der Manipulation dieser Adresse über Basic vorsichtig sein, da ein Verfälschen dieses Wertes manchen Basic-Befehlen ernsthafte Schwierigkeiten bereiten kann.

– *Adressen 178/179: Anfangsadresse des Kassettenpuffers*

Wie beim C64. Eine interessante Anwendung dieser Adressen wird bei »Adressen 828-1019: Kassettenpuffer« beschrieben.

– *Adressen 183-187: Informationen über das aktuelle File*

Diese Adressen, die bei C64 und C128 die gleiche Funktion haben, beinhalten folgende Informationen:

183:	Länge des aktuellen Filenamens
184:	logische Filenummer der aktuellen Datei
185:	aktuelle Sekundäradresse
186:	aktuelle Gerätenummer
187/188:	PEEK(187)+256*PEEK(188) liefert die Adresse, ab welcher der Name des aktuellen Files im Speicher liegt.

– *Adresse 198: Anzahl der Zeichen im Tastaturpuffer*

Mit »POKE 198,0« kann beim C64 der Tastaturpuffer gelöscht werden, d.h. alle vor diesem Befehl gedrückten, aber noch nicht über GET oder INPUT verarbeiteten Eingaben werden »vergessen«.
Andere C64-Anwendung: »WAIT198,1« wartet auf einen Tastendruck.
Beim C128 wird diese Aufgabe von der Adresse 208 übernommen. Allerdings sollte auf die genannten POKE- und WAIT-Anwendungen verzichtet werden, indem man sich der Basic-7.0-Befehle bedient:

```
DO:GET A$:LOOP WHILE A$<>"" entspricht POKE 198 (C128:208),0
GETKEY A$                   entspricht WAIT 198,1:GET A$
```

Um den Tastaturpuffer sinnvoll anzuwenden, wie es in 3.7.5 besprochen wird, benötigt man aber diese Adresse 208.

- *Adresse 203: Code der gedrückten Taste*

Mit PEEK(203) kann man beim C64 den Code der gerade gedrückten Taste ermitteln (64= keine Taste gedrückt). Dieser Code ist übrigens weder mit dem ASCII- noch mit dem Bildschirmcode identisch, sondern stellt einen Tastatur-Zwischencode dar.
Beim C128 ist 212 die zuständige Adresse. Zusätzlich liefert PEEK(213) den entsprechenden Code der letzten gedrückten Taste, wofür beim C64 keine Adresse vorhanden ist.

- *Adresse 204: Cursor an/aus (0 = Cursor blinkt)*

Beim C64 kann mit »POKE 204,0« der Cursor auf einfache Weise angeschaltet werden. Dadurch kann er auch außerhalb von INPUT als Aufforderung zur Eingabe (z.B. Tastendruck) verwendet werden.
Mit »POKE 207,0:POKE 204,1« wird er wieder ausgeschaltet.
Um diese Befehle auf den C128 umzuschreiben, verwendet man nicht POKE, sondern SYS (und BANK, wenn nicht Bank 15 eingestellt ist):

```
BANK15:SYS DEC("CD6F")    Cursor einschalten
BANK15:SYS DEC("CD9F")    Cursor ausschalten
```

Der Cursor blinkt immer an der Stelle, an der die nächste Bildschirmausgabe erfolgen würde. Befindet sich an dieser Position schon ein Zeichen, so blinkt dieses mit dem Cursor. Beispiel:

```
100 PRINT "'*' BLINKT: *";CHR$(157);:REM 157 = CRSR LEFT
110 BANK 15:SYS DEC("CD6F"):REM Cursor ein
120 GETKEY A$:REM Auf Tastendruck warten
130 SYS DEC("CD9F"):REM Cursor aus
```

Die Einstellung von BANK 15 ist beim zweiten SYS-Befehl (Zeile 130) nicht mehr nötig, da dort die Einstellung von Zeile 110 immer noch Gültigkeit hat.

- *Adressen 211/214: Cursorpositionierung*

Da der C64 über den Befehl »PRINT AT« (PRINT an beliebige Bildschirmposition) nicht verfügt, wird dieser oft über POKEs in die Adressen 211 und 214 mit anschließendem PRINT simuliert.
Beim C128 ist dies nicht nötig, dort steht der Befehl CHAR zur Verfügung (Befehlserklärung in 3.4.5).

- *Adressen 512-600: Eingabepuffer*

Diese Adressen sind bei C64 und C128 gleich, allerdings ist der C128-Eingabepuffer größer: er reicht von 512 bis Adresse 673, damit er die 160 Zeichen, die pro Eingabe erlaubt sind, aufnehmen kann (der Puffer hätte sogar Platz für mehr Zeichen).
Mit Hilfe dieses Eingabepuffers sind wir in der Lage, eine kleine Unterroutine zu programmieren, die dem INPUT-Befehl entspricht, aber alle Zeichen (also auch »:«,»,« und »;«) verarbeiten kann:

```
50000 REM Unterprogramm mit "GOSUB 50000" aufrufen
50010 :
```

```
50020 BANK 15:SYS DEC("4F93"):REM Eingabe in Puffer holen
50030 AD=512
50040 XX$=""
50050 DO WHILE PEEK(AD)<>0
50060 :XX$=XX$+CHR$(PEEK(AD))
50070 :AD=AD+1
50080 LOOP
50090 RETURN
```

In XX$ steht danach die Eingabe. Vor dieser wird, im Gegensatz zum normalen INPUT, kein Fragezeichen ausgegeben.
Als Fehlermeldung tritt »?STRING TOO LONG« auf, wenn die Eingabe länger als 160 Zeichen ist.
Unser Unterprogramm läßt sich sogar noch in bezug auf Geschwindigkeit und Speicherverbrauch erheblich optimieren:

```
50000 REM Unterprogramm mit "GOSUB 50000" aufrufen
50010 BANK15:SYSDEC("4F93"):AD=512:XX$="":DOWHILEPEEK(AD):
          XX$=XX$+CHR$(PEEK(AD)):AD=AD+1:LOOP:RETURN
```

Dieses Unterprogramm befindet sich nicht auf der Programmdiskette, da es sehr kurz ist und somit bei jedem Programm neu eingetippt werden kann. Auch bei schon vorhandenen Programmen ist es sicher oft nützlich, den Eingabekomfort mit Hilfe dieser Routine erheblich zu steigern. Eine Eingaberoutine, die professionellen Ansprüchen gerecht wird und noch wesentlich ausgefeilter ist als diese hier, wird in 3.6.4 entwickelt.

- *Adressen 631-640: Tastaturpuffer*

Der Tastaturpuffer liegt beim C128 im Bereich 842-851. Diese Adressen sind bei der Anpassung von C64-Tastaturpufferanwendungen einzusetzen (siehe auch »Adresse 198: Anzahl der Zeichen im Tastaturpuffer«).
Wie man den Tastaturpuffer des C128 sinnvoll nutzbar macht, wird in 3.7.5 gezeigt.

- *Adresse 646: aktueller Farbcode»*

Diese Adresse wurde schon unter a) besprochen. Siehe »PEEKs und POKEs für Bildschirmfarben«.

- *Adresse 649: Größe des Tastaturpuffers*

Da Adresse 649 die maximale Größe des Tastaturpuffers (bis zu 10) enthält, schaltet

```
POKE 649,0
```

die Tastatur des C64 ab.
Beim C128: POKE 2592,0

- *Adresse 650: Tastaturwiederholung (Repeat)*

Tastaturwiederholung heißt, daß man eine Taste gedrückt halten kann, anstatt Sie mehrfach drücken zu müssen.

Da der C64 im Einschaltzustand über die Tastaturwiederholung (Repeat) nicht verfügt (Ausnahmen: Leertaste, Cursortasten, <INST/DEL>), kann dies über POKE eingestellt werden:

```
POKE 650,0   Repeat für <SPACE>, Cursortasten, <INST/DEL)
POKE 650,64  kein Repeat
POKE 650,128 Repeat für alle Tasten (C128-Einschaltzustand)
```

Beim C128 ist »Repeat für alle Tasten« voreingestellt; folglich muß dies nicht erst mit POKE bewirkt werden. Soll aber die Wiederholfunktion teilweise (POKE...,0) oder ganz (POKE...,64) abgeschaltet werden, so muß statt 650 beim C128 die Adresse 2594 eingesetzt werden. Es gelten die gleichen Werte (0,64 und 128).

- *Adresse 651: Zähler für Wiederholungsgeschwindigkeit*

Diese Adresse muß beim C128 durch »2595« ersetzt werden. Sie entscheidet darüber, wie schnell die Repeat-Funktion abläuft (siehe »Adresse 650: Tastaturwiederholung (Repeat)«.

- *Adresse 652: Zähler für Wiederholungsverzögerung*

Auch diese Adresse steht in Verbindung mit der Tastaturwiederholung. Die Adresse 2596 übernimmt deren Aufgabe beim C128.

- *Adresse 653: Flag für <SHIFT>, <CBM> und <CONTROL>*

Diese Adresse wird beim C128 durch 211 ersetzt und sagt aus, ob <SHIFT>, <CBM>, <CONTROL> oder eine Kombination aus diesen gedrückt wurde. Dabei gilt Tabelle 3.8 für PEEK(211) bzw. PEEK(653) beim C64.

Tabelle 3.8: *Mögliche Kombinationen der C64-Speicherzelle 653*

0	keine der drei Tasten <SHIFT>, <CBM> und <CONTROL>
1	<SHIFT>
2	<CBM>
3	<SHIFT>+<CBM>
4	<CONTROL>
5	<SHIFT>+<CONTROL>
6	<CBM>+<CONTROL>
7	<SHIFT>+<CBM>+<CONTROL>

Zusätzlich fragt der C128 noch <ALT> und <CAPS LOCK> mt dieser Adresse ab, was PEEK(653) auf einem C64 selbstverständlich nicht kann (auch nicht im C64-Modus des C128), wie Tabelle 3.9 zeigt:

Tabelle 3.9: *Mögliche Zusatzkombinationen der C128-Adresse 211 (siehe auch Tabelle 3.8)*

8	<ALT>
9	<ALT>+<SHIFT>
10	<ALT>+<CBM>
11	<ALT>+<SHIFT>+<CBM>
12	<ALT>+<CONTROL>
13	<ALT>+<SHIFT>+<CONTROL>
14	<ALT>+<CBM>+<CONTROL>
15	<ALT>+<SHIFT>+<CBM>+<CONTROL>
16	<CAPS LOCK>
17	<CAPS LOCK>+<SHIFT>
18	<CAPS LOCK>+<CBM>
19	<CAPS LOCK>+<SHIFT>+<CBM>
20	<CAPS LOCK>+<CONTROL>
21	<CAPS LOCK>+<SHIFT>+<CONTROL>
22	<CAPS LOCK>+<CBM>+<CONTROL>
23	<CAPS LOCK>+<SHIFT>+<CBM>+<CONTROL>
24	<CAPS LOCK>+<ALT>
25	<CAPS LOCK>+<ALT>+<SHIFT>
26	<CAPS LOCK>+<ALT>+<CBM>
27	<CAPS LOCK>+<ALT>+<SHIFT>+<CBM>
28	<CAPS LOCK>+<ALT>+<CONTROL>
29	<CAPS LOCK>+<ALT>+<SHIFT>+<CONTROL>
30	<CAPS LOCK>+<ALT>+<CBM>+<CONTROL>
31	<CAPS LOCK>+<ALT>+<SHIFT>+<CBM>+<CONTROL

Folgende Eingabe und ein wenig Fingerfertigkeit ermöglichen es Ihnen, diese Codewerte nachzuvollziehen:

```
DO:PRINT PEEK(211):LOOP
```

Wenn bestimmte Tasten ausgesondert werden sollen, sind folgende PEEK-Funktionen zu verwenden (Tabelle 3.10):

Tabelle 3.10: *Aussonderungsmöglichkeit mit dem AND-Befehl*

PEEK(211) AND 1	<SHIFT>
PEEK(211) AND 2	<CBM>
PEEK(211) AND 4	<CONTROL>
PEEK(211) AND 8	<ALT>
PEEK(211) AND 16	<CAPS LOCK>

Diese Funktionen liefern den Wert 1, wenn die entsprechende Taste gedrückt ist, andernfalls 0.

Adressen 768-771: Programmschutz

Damit ein Programm nach seinem Beenden (über Fehlermeldung, <RUN/STOP>+ <RESTORE> oder END) einen Reset auslöst und somit zumindest teilweise verlorengeht, läßt man beim C64 folgende Befehle ausführen:

```
P768,226:POKE 769,252:POKE 770,226:POKE 771,252
```

Da der Reset-Einsprung des C128 mit 65341 ein anderer als 64738 (C64) ist, müssen auch die POKEs angepaßt werden (die Adressen bleiben):

```
POKE 768, 61:POKE 769,255:POKE 770, 61:POKE 771,255
```

Eleganter ist jedoch beim C128 der Einsatz der programmgesteuerten Fehlerbehandlung mit den Befehlen TRAP und RESUME; eine Erklärung dazu finden Sie in Kapitel 3.4.6.

– *Adressen 780-783: Kommunikation zwischen Basic und Maschinensprache*

Damit ein Basic-Programm an eine Maschinenroutine Werte übergeben kann, werden diese in den Adressen 780-783 gespeichert und können nach dem SYS-Befehl über PEEK ausgelesen werden:

```
780 Akkumulator
781 X-Register
782 Y-Register
783 Prozessorstatus (Statusregister)
```

Beim C128 stehen diese Werte in den Adressen 6-9 in gleicher Reihenfolge im Speicher, aber die erweiterte Syntax von SYS (siehe 3.2) und der RREG-Befehl (siehe 3.4.6) sind gleichermaßen geeignet.

– *Adressen 792/793: Noch einmal – Programmschutz*

Der Befehl

```
POKE 792,193
```

schaltet beim C64 die Kombination <RUN/STOP>+<RESTORE> ab. Die <RUN/STOP>-Taste für sich allein funktioniert aber weiterhin. Dies gilt auch für die C128-Entsprechung

```
POKE 792, 51:POKE 793,255
```

– *Adresse 808: Schon wieder - Programmschutz*

Ein recht weitverbreiteter Trick für den C64 ist das Abschalten der <RUN/STOP>- und der <RESTORE>-Taste mit

```
POKE 808,225
```

Diese Programmschutzmaßnahme erledigt beim C128 der Befehl

```
POKE 808, 98
```

Der Nachteil, daß damit die interne Uhr für Basic (TI,TI$) unbrauchbar wird, besteht auch beim C128. Daß durch den jeweiligen POKE bei LIST nur »Zeichenmüll« erscheint, ist aber nur beim C64 so.

– *Adressen 828-1019: Kassettenpuffer*

Bei Kassettenoperationen wird dieser Speicherbereich beim C64 als Zwischenspeicher eingesetzt; solange die Datassette nicht verwendet wird, können dort Sprites oder Maschinenprogramme aufbewahrt werden (bei C64-Programmen ein beliebtes Verfahren).
Beim C128 liegt der Kassettenpuffer im Bereich 2816-3071 (hexadezimal: $0b00-$0bff). Sprites können dort allerdings nicht mehr ohne weiteres abgelegt werden; für diesen Zweck ist der Speicherbereich 3584-4095 reserviert.
Maschinenprogramme, die im Kassettenpuffer stehen, sind dort unter den gleichen Bedingungen wie beim C64 untergebracht. Um den Kassettenpuffer in den 40-Zeichen-Bildschirmspeicher zu legen (!) und dadurch den Bereich 2816-3071 (ursprüngliche Lage des Kassettenpuffers) zu schützen, sind folgende POKEs nötig:

```
POKE 178,0:POKE 179,4
```

Diese POKE-Befehle gelten auch für den C64.
Nach Kassettenoperationen muß, wenn die genannten POKE-Befehle eingesetzt wurden, der 40-Zeichen-Bildschirm mit SCNCLR(0) gelöscht werden, da dort sonst der Inhalt des Kassettenpuffers steht, der am 40er-Bildschirm für wildes Chaos sorgt.

– *Adressen 2040-2047: Zeiger für Sprites*

Auch beim C128 sind die Sprite-Zeiger im Bereich 2040-2047 untergebracht. Diese werden vorbelegt, damit die Sprites 1-8 im für Basic vorgesehenen Sprite-Speicher 3584-4095 liegen. Aufgrund der Voreinstellung von 2040-2047 durch den Basic-Interpreter bei der Initialisierung müssen die Sprite-Zeiger nicht gesetzt werden, solange die Sprites in Basic 7.0 programmiert werden.

– *Adressen 2048-40959: Basic-Benutzerspeicher*

Der Basic-Benutzerspeicher ist beim C128 aufgeteilt (siehe 3.2):

```
Bank 0, Adressen 7182-65279: Speicherbereich für Programm
Bank 1, Adressen 1024-65279: Speicherbereich für Variablen
```

Diese andere Speicherorganisation hat auch Auswirkungen auf die FRE-Funktion, was in 3.2 erläutert wird.

3.5.5 Simon's-Basic-Befehle umschreiben

Aufgrund der Beliebtheit und großen Verbreitung der Erweiterung Simon's Basic zum C64 wurden viele Simon's-Basic-Programme veröffentlicht, z.B. in Computerzeitschriften, wobei 64'er und auch Happy-Computer eine Reihe erstklassiger Programme als Listings abgedruckt haben, die es sicher wert sind, umgeschrieben zu werden.

Deshalb sollen in diesem Unterkapitel alle Simon's-Basic-Befehle aufgeführt werden, und sofern ein Ersetzen durch Basic-7.0-Anweisungen mit vertretbarem Aufwand möglich ist, wird dies auch hier erklärt.
Andere Basic-Erweiterungen zum C64 können leider nicht berücksichtigt werden, da sich damit ein ganzes Buch füllen ließe; ich habe das gängigste Produkt dieser Art ausgewählt und hoffe, daß Sie mit dieser Wahl einverstanden sind.
Voraussetzung für das Umschreiben ist übrigens, daß Sie über Simon's Basic oder zumindest die dazugehörige Anleitung und Simon's-Basic-Kenntnisse verfügen, da hier unmöglich die Befehlsbeschreibung wiederholt werden kann.
Wichtiger Hinweis: Simon's-Basic-Programme können zwar in den C128 geladen werden, allerdings sind die Zusatzbefehle aufgrund der unterschiedlichen Codierungsverfahren von Simon's Basic und Basic 7.0 nicht über LIST erkennbar.
Deshalb sollte man ein Simon's-Basic-Programm zunächst im C64-Modus mit Simon's Basic ausdrucken und dann das Umschreiben durchführen. Mit Hilfe des Simon's-Basic-Befehls OPTION können dabei die Simon's-Basic-Zusatzbefehle hervorgehoben werden.
Die Befehle von Simon's Basic unterteilen wir in folgende elf Gruppen:

- Programmierhilfen
- Befehle zur strukturierten Programmierung und Fehlerbehandlung
- Befehle zur hochauflösenden und Multicolor-Grafik
- Sprite-Befehle
- Sound-Befehle
- String-Befehle
- Funktionen zur Zahlenverarbeitung
- Befehle zur Bildschirmsteuerung
- Befehle zur Abfrage von Joystick, Drehregler und Lichtgriffel
- Sonstige Befehle
- Befehle, die nicht im Simon's-Basic-Handbuch stehen

Nun kommen wir zur Behandlung des Umschreibens.

– Programmierhilfen

AUTO – automatische Zeilennumerierung

Bei Simon's Basic wird zusätzlich zur Schrittweite noch die Startzeile angegeben (»AUTO 100,10«); beim C128 wird die Startzeile nicht übermittelt, da die automatische Zeilennumerierung erst nach Eingabe einer Zeile einsetzt (»AUTO 10«).

COLD – Kaltstart

Um das Aus- und wieder Einschalten des Computers zu ersparen, löst COLD einen Software-Reset (Kaltstart) aus. In Basic 7.0 ist hierfür kein eigener Befehl vorhanden, aber man kann folgende Hilfskonstruktion verwenden, die gleiche Wirkung hat:

```
BANK15:SYS65341
```

DELAY - Verzögerung bei LIST einstellen

Dies ist in Basic 7.0 nicht möglich.

DISAPA - Maßnahme zum Programmschutz

In Simon's Basic kann dieser Befehl einen einfachen Schutz gegen LIST ermöglichen. Basic 7.0 sieht hierfür keinen Befehl vor. Da aber der DISAPA-Schutz von Simon's Basic auf leichte Weise mit Hilfe eines Programms, wie es im »Commodore 64-Buch, Band 5: Ein Leitfaden durch Simon's Basic« von Markt & Technik vorgestellt wird, rückgängig gemacht werden kann, ist das Fehlen eines solchen Befehls kein schmerzlicher Verlust.

DISPLAY oder KEY0 - Funktionstastenbelegung anzeigen

In Basic 7.0 schreibt man hierfür

```
KEY
```

Die Ausgabe der Funktionstastenbelegung erfolgt auf verblüffend ähnliche Art und Weise: Auch in Basic 7.0 wird vor einer Belegung der KEY-Befehl mit der richtigen Nummer und dem Komma ausgegeben, damit die Belegung leicht geändert werden kann.

DUMP - Variablenwerte anzeigen

Da es recht mühsam ist, immer nach einer Programmunterbrechung (über den Befehl STOP oder die gleichnamige Taste) einen oder mehrere PRINT-Befehle einzugeben, um die benötigten Variablen anzuzeigen, stellt Simon's Basic hierfür den Befehl »DUMP« zur Verfügung. Ein entsprechender Befehl ist in Basic 7.0 nicht vorhanden. Tip: PRINT-Befehl mit allen auszugebenden Variablen auf eine Funktionstaste legen, wenn die Variablenausgabe häufig nötig ist. Beispiel:

```
PRINT AB,AC,D,I,H%,I%,JJ%,KL%,A$,B$,G$,JN$,K(0),K(1),K(2),K(3)
```

auf Funktionstaste F1 legen (mittels KEY).

FIND - Programmstellen suchen lassen

Mit FIND kann in Simon's Basic ein Text (Befehl, Variable, String) innerhalb eines Programms gesucht werden, was die lästige Suche über LIST überflüssig macht; die Zeilen, in denen der gesuchte Text vorkommt, werden ausgegeben.
Im Standard-Basic des C128 ist ein ähnlicher Befehl nicht vorhanden. In 3.7.3 wird aber eine entsprechende Routine vorgestellt, die den FIND-Befehl von Simon's Basic übertrifft:
- die gefundenen Zeilen werden gelistet
- die Suche innerhalb eines bestimmten Zeilenbereichs ist möglich.

KEY - Funktionstasten belegen

Dieser Befehl hat beim C128 dieselbe Syntax wie in Simon's Basic, wenn man von der Variante »KEY0« absieht (siehe »DISPLAY oder KEY0 - Funktionstastenbelegung anzeigen«).

MERGE - Programme koppeln

Wie FIND, ist auch dieser Befehl beim C128 nicht standardmäßig vorhanden, aber in 3.7.4 wird eine Hilfslösung vorgestellt, die in der Wirkung dem MERGE-Kommando von Simon's Basic haargenau entspricht.

OLD - Programm nach NEW oder RESET retten

Dieser Befehl ist, wie FIND und MERGE, in Basic 7.0 nicht implementiert. Auch für diese wichtige Funktion wird eine Ersatzlösung vorgestellt (3.7.2).

OPTION - Simon's-Basic-Befehle beim Listen hervorheben

Dieser Befehl würde in Basic 7.0 keinen Sinn ergeben und ist dementsprechend auch nicht eingebaut.
Soll jedoch ein Simon's-Basic-Programm umgeschrieben werden (beispielsweise auf das Basic 7.0 des C128), ist dieser Befehl recht nützlich, da Sie beim Listen mit Simon's Basic sofort ermitteln können, welche Befehle umgeschrieben werden müssen.

PAGE - Scrolling bei LIST verhindern/Bildschirmbereich einstellen

Das Scrolling wird mit ESC-M unterdrückt und durch ESC-L wieder zugelassen; einen Bildschirmbereich kann man mit WINDOW einstellen (oder ESC-B und ESC-T im Eingabemodus).

RENUMBER - Zeilennummern umnumerieren

Dieser Befehl trägt auch auf dem C128 die Bezeichnung »RENUMBER« und kann die von Simon's Basic gewohnte Syntax verarbeiten, allerdings bietet Basic 7.0 noch weitere Möglichkeiten:
- GOTO, GOSUB usw. werden auch umnumeriert
- Programm kann auch teilweise umnumeriert werden

SECURE - Ergänzung zu DISAPA

Dieser Befehl bewirkt nur das eigentliche Schützen der durch DISAPA gekennzeichneten Befehle und ist, wie DISAPA, auf dem C128 nicht implementiert.

TRACE/RETRACE - Programmablaufverfolgung

Auf dem C128 wird die Programmablaufverfolgung durch TRON eingeschaltet (entspricht »TRACE x« mit x <>0) und mit TROFF ausgestellt (entspricht »TRACE 0«). Da die jeweilige Zeilennummer mitten im Text und nicht in einem Bildschirmfenster dargestellt wird, gibt es kein RETRACE-Äquivalent in Basic 7.0.

– Befehle zur strukturierten Programmierung und Fehlerbehandlung

ELSE – Erweiterung für den IF-Befehl

Dieser Befehl entspricht dem gleichnamigen Basic-7.0-Kommando; während aber ELSE in Simon's Basic vom vorhergehenden Befehl nicht durch einen Doppelpunkt abgegrenzt werden muß, erfordert Basic 7.0 diese Syntax. Beispiel:

Simon's Basic: IF A>B THEN PRINT "A>B" ELSE PRINT "A<B ODER A=B"
Basic 7.0: IF A>B THEN PRINT "A>B":ELSE PRINT "A<B ODER A=B"

RCOMP – letzte IF-Bedingung abfragen

Dieser Befehl wird von Simon's Basic zur Verfügung gestellt, damit eine blockweise Bearbeitung in verschiedenen Zeilen der THEN-/ELSE-Teile erfolgen kann. In Basic 7.0 wird dies durch eine andere (und bessere) Lösung ermöglicht: BEGIN...BEND

REPEAT...UNTIL – Schleifenkonstruktion

REPEAT wird in Basic 7.0 durch DO ersetzt, statt UNTIL muß LOOP UNTIL geschrieben werden.

Simon's Basic: REPEAT:A=A+1:UNTIL A=10
Basic 7.0: DO:A=A+1:LOOP UNTIL A=10

LOOP...EXIT IF...END LOOP – weitere Schleifenkonstruktion

LOOP muß durch DO, »EXIT IF ...« durch »IF...THEN EXIT« und schließlich END LOOP durch LOOP ersetzt werden:

Simon's Basic:
```
100 LOOP
110 :A=A+1
120 :EXIT IF A=10
130 END LOOP
```

Basic 7.0:
```
100 DO
110 :A=A+1
120 :IF A=10 THEN EXIT
130 LOOP
```

Zusätzlich wäre in Basic 7.0 auch die Verwendung von WHILE oder UNTIL bei DO oder LOOP möglich.

PROC, END PROC, CALL und EXEC – Prozeduren mit Struktur

Diese Befehle können auf zweierlei Arten in Basic 7.0 ersetzt werden. Entweder setzt man Zeilennummern statt Marken ein und handhabt die ehemaligen Prozeduren als Unterprogramme und arbeitet zeilenorientiert, oder man verwendet die in 3.4.2 vorgestellte Routine

»LABEL 128«. Bei der zweiten Lösung muß der PROC-Befehl durch REM (die Bezeichnung der Prozedur muß als Label erhalten bleiben und unmittelbar auf REM folgen), END PROC durch RETURN, CALL durch GOTO und EXEC durch GOSUB ersetzt werden.

Simon's Basic:
```
100 EXEC DRUCKROUTINE
110 ...
120 CALL DRUCKROUTINE
...
10000 PROC DRUCKROUTINE
10010 PRINT ...
...
10500 END PROC
```

Basic 7.0 mit
»LABEL 128«:
```
100 GOSUB "DRUCKROUTINE"
110 ...
120 GOTO "DRUCKROUTINE"
...
10000 REMDRUCKROUTINE
10010 PRINT ...
...
10500 RETURN
```

LOCAL/GLOBAL – lokale und globale Variablen

Diese Möglichkeit bietet Basic 7.0 nicht. Man muß hierfür Hilfsvariablen verwenden, die die globalen Werte von Variablen retten.

ON ERROR – Fehlerbehandlungsroutine einschalten

In Basic 7.0 lautet ein ähnlicher Befehl »TRAP«.

ERRN und ERRLN – Fehlernummer und Fehlerzeile

»ERRN« wird in Basic 7.0 durch ER«,
»ERRLN« durch »EL« umschrieben. Die Bedeutung ändert sich aber nicht.

OUT – Standardfehlermeldung ausgeben

Das Abstellen der programmierten Fehlerbehandlung geschieht in Basic 7.0 ebenfalls über den TRAP-Befehl: Zum Abschalten der Fehlerbehandlung wird keine Zeilennummer angegeben.

– Befehle zur hochauflösenden und Multicolor-Grafik

HIRES, MULTI, COLOUR und CSET – Grafikmodus wählen/Farben setzen

In Basic 7.0 werden hierfür GRAPHIC und COLOR verwendet.

LOW COL und HI COL - Zusatzfarben für Multicolor

Ähnliches kann über COLOR in Basic 7.0 bewirkt werden.

PLOT - Punkt setzen/löschen

Statt »PLOT x,y,Farbquelle« schreibt man in Basic 7.0:

```
DRAW Farbquelle,x,y
```

Beispiel: "DRAW 1,100,150" statt "PLOT 100,150,1"

Man muß beachten, daß die Angabe der Farbquelle beim C128 andere Werte erfordert als in Simon's Basic; 0 und 1 (löschen und setzen) stimmen überein, aber das Invertieren ist in Basic 7.0 nicht möglich. Im Multicolor-Modus stimmen sogar noch 2 (Multicolorfarbe 1) und 3 (Multicolorfarbe 2) überein.

LINE - Linie ziehen

Statt »LINE x1,y1,x2,y2,Farbquelle« schreibt man in Basic 7.0:

```
DRAW Farbquelle,x1,y1 TO x2,y2
```

Beispiel: "DRAW 1,10,20 TO 150,100" statt "LINE 10,20,150,100,1"

Für die Farbquelle gilt das bei PLOT Gesagte.

REC - Rechteck zeichnen

In Basic 7.0 erledigt dies der BOX-Befehl.

CIRCLE - Kreis oder Ellipse zeichnen

In Basic 7.0 wird dafür der gleichnamige Befehl verwendet, der aber eine andere Syntax hat und außerdem Polygone (Vielecke) zeichnen kann, was CIRCLE in Simon's Basic nicht ermöglicht.

ARC - Segment zeichnen

Dies wird in Basic 7.0 ebenfalls mit CIRCLE bewerkstelligt.

ANGL - Radien zeichnen

Dieser Befehl ist in Basic 7.0 zwar eigentlich nicht vorhanden, aber die Möglichkeit, bei DRAW Winkel und Abstand anzugeben, ermöglicht eine Hilfskonstruktion:

```
DRAW 1,100,100 TO 50;70
```

zeichnet vom Punkt 100,100 im Winkel von 50 Grad (nach der Kompaßrose, siehe 3.4.3) einen Radius von 70 Punkten Länge.
Auf diese Weise kann die Simon's-Basic-Anweisung ANGL in den allermeisten Fällen auf den C128 übertragen werden.

BLOCK – ausgefülltes Rechteck zeichnen

In Basic 7.0 fällt dies unter den BOX-Befehl; hängt man an die Koordinatenangabe »,1« an, wird das von BOX gezeichnete Rechteck automatisch ausgefüllt.

PAINT – Fläche ausfüllen

In Basic 7.0 heißt der entsprechende Befehl auch »PAINT«, hat aber eine etwas andere Syntax.

ROT und DRAW – Figur aus Linien zusammensetzen und drehen

Entsprechende Befehle kennt der C128 nicht, allerdings sind die vielfältigen Möglichkeiten des C128-DRAW-Befehls als Ersatz geeignet (siehe 3.4.3), da dieser relative Koordinatenangaben zuläßt, welche Ähnliches bewirken können.

CHAR und TEXT – Text in die hochauflösende Grafik schreiben

Dies wird in Basic 7.0 durch den gleichlautenden Befehl »CHAR« (andere Syntax beachten!) erledigt, der allerdings den Befehlen TEXT und CHAR von Simon's Basic nachsteht:
- beim C128 keine Auswahl der Zeichengröße möglich
- Position nur nach Zeilen und Spalten wählbar, aber nicht als Koordinatenangabe für Grafik

TEST – Punkt testen (Farbquelle ermitteln)

In Basic 7.0 muß hierfür die RDOT-Funktion verwendet werden, wobei LOCATE zur Angabe der Koordinaten vorausgehen sollte.

– Sprite-Befehle

DESIGN – Spritemuster definieren

Dieser Befehl, dem eine Reihe von Zeilen, die mit »§« beginnen, folgen muß, fehlt in Basic 7.0 leider; dafür ist mit »SPRDEF« ein Sprite-Editor erreichbar.

CMOB – Farben für Multicolor-Sprites setzen

In Basic 7.0 wird dies von SPRCOLOR bewirkt.

MOB SET – Sprite-Attribute setzen

Die Entsprechung in Basic 7.0 ist SPRITE, hat aber eine andere Syntax.

MMOB und RLOCMOB – Sprite positionieren bzw. bewegen

Hierfür dient beim C128 ein einziger Befehl: MOVSPR.
Achtung: MOVSPR schaltet ein Sprite nicht an; dies muß über »SPRITE« geschehen.

MOB OFF - Sprite ausschalten

Dies wird in Basic 7.0 durch »SPRITE« erledigt.

DETECT und CHECK - Sprite-Kollision vorbereiten bzw. abfragen

In Basic 7.0 wird mit COLLISION die Sprite-Kollisionsabfrage vorbereitet und über die BUMP-Funktion der Kollisionstyp ermittelt.

- Sound-Befehle

PLAY und MUSIC - Musik-String abspielen

Der gleichnamige Basic-7.0-Befehl arbeitet ähnlich. Die Syntax ist aber anders.

VOL - Lautstärke regulieren

Entspricht exakt (in Wirkung und Syntax) dem C128-Befehl VOL.

WAVE - Wellenform definieren

In Basic 7.0 geschieht dies über »ENVELOPE«

ENVELOPE - Hüllkurve definieren

Kann durch den gleichnamigen Basic-7.0-Befehl ersetzt werden, der allerdings eine erweiterte Syntax besitzt.

- String-Befehle

PLACE - Teilstring in String suchen

Dies erledigt beim C128 die Funktion INSTR, die eine andere Syntax hat (unbedingt beachten!), denn Teil- und Hauptstring werden in anderer Reihenfolge angegeben.

USE - Formatierte Zahlenausgabe

Dazu verwendet man in Basic 7.0 die Anweisung PRINT USING.

CENTRE - String in Zeilenmitte ausgeben

In Basic 7.0 kann man dies über PRINT USING bewirken, wofür das Formatsteuerzeichen »=« vorgesehen ist.
Man muß auch ein Formatfeld von 40 oder 80 Zeichen Länge definieren, da PRINT USING nur innerhalb eines Formatfeldes zentriert. Die zentrierte Ausgabe ist aber auch mit TAB möglich:

```
PRINT TAB(40-LEN(A$)/2);A$   im 40-Zeichen-Modus
PRINT TAB(80-LEN(A$)/2);A$   im 80-Zeichen-Modus
```

DUP - String vervielfachen

Dies ist in Basic 7.0 nur mit Hilfe einer Additionsschleife möglich. Beispiel:

Simon's Basic: A$=DUP("***",10)
Basic 7.0: A$="":FOR I=1 TO 10:A$=A$+"***":NEXT

- Funktionen zur Zahlenverarbeitung

EXOR - Exklusiv-Oder (Entweder-Oder) für Bitverknüpfungen

In Basic 7.0 heißt diese Funktion »XOR«.

MOD, DIV und FRAC - mathematische Funktionen

MOD bildet die im mathematischen Gebrauch definierte Modulo-Funktion (Rest bei Division), DIV ist eine Division ohne Rest und FRAC zieht Nachkommastellen einer Dezimalzahl heraus. In Basic 7.0 muß man dies mit DEF FN definieren:
Dabei wird der jeweiligen Funktion die Zahl als Argument übergeben, in der Variablen T muß der Teiler (Divisor) stehen:

```
DEF FN MOD(Z) =INT(((Z/T-INT(Z/T))*T)+.5)
DEF FN DIV(Z) =INT(Z/T)
DEF FN FRAC(Z)=Z/T-INT(Z/T)
```

% und $ - Binär- und Hexadezimalsystem

Das Binärsystem wird von Basic 7.0 zwar nicht beherrscht, aber das Hexadezimalsystem kann mit DEC ins Dezimalformat gewandelt werden. Beispiel:

Simon's Basic: $D400
Basic 7.0: DEC("D400")

Binärzahlen kann man auf folgende Weise umrechnen:
1) <F8> drücken oder »MONITOR« eingeben
2) Binärzahl wie in Simon's Basic angebem (z.B. %1001100) und <RETURN> drücken
3) Das Ergebnis erscheint in vier Zahlensystemen; vor der Dezimalzahl steht »+«
4) <X><RETURN> eingeben, um ins Basic zu gelangen
Dann kann der %-Befehl ersetzt werden, wenn schon keine Entsprechung in Basic 7.0 vorhanden ist.

- Befehle zur Bildschirmsteuerung

Folgende Befehle von Simon's Basic sind in Basic 7.0 nicht ohne weiteres zu ersetzen:
FLASH, OFF, BFLASH, BFLASH 0, FCHR, FILL, INV, MOVE,
LEFTB, RIGHTB, LEFTW, RIGHTW, UPW, DOWNW

FCOL - Bildschirmbereich mit Farbe füllen

Dazu kann der WINDOW-Befehl verwendet werden, der das definierte Window automatisch mit der aktuellen Zeichenfarbe, die über »COLOR 5,Farbcode« geändert wird, auffüllt.
Mit PRINTCHR$(19)CHR$(19) kann man das Window auflösen.

UPB und DOWNB - einfacher Ersatz mit ESC-Sequenzen

ESC-V bewirkt Scrolling nach oben (UPB)
ESC-W bewirkt Scrolling nach unten (DOWNB)

```
PRINT CHR$(27)"V"     statt:   UPB
PRINT CHR$(27)"W"     statt:   DOWNB
```

SCRSV und SCRLD - Bildschirm speichern bzw. laden

Dies geht beim C128 nur mit dem 40-Zeichen-Bildschirm:

```
BSAVE "SCREEN",ON B0,P1024 TO P2023 speichern
BLOAD "SCREEN",ON B0                laden
```

COPY und HRDCPY - Ausgabe des Bildschirms auf den Drucker

In 3.8 werden wir Hardcopy-Routinen für den C128 entwickeln (in Basic und Maschinensprache), die diese Befehle ersetzen können.
Basic 7.0 kennt zu diesem Zweck keine Kommandos.

- Befehle zur Abfrage von Joystick, Drehregler und Lichtgriffel

Das Umschreiben dieser Befehle auf den C128 ist unproblematisch:

JOY(n)	fragt Joystick in Port »n« ab
POT(n)	fragt Drehregler »n« (1-4) ab
PEN(..)	liefert Informationen über den Lichtgriffel

Diese Befehle sind auf dem C128 teilweise mächtiger als ihre Simon's-Basic-Äquivalente und haben eine andere Syntax (insbesondere der PEN-Befehl).

- Sonstige Befehle

AT und PRINT AT - Cursorpositionierung

In Basic 7.0 wird mit dem CHAR-Befehl die Textausgabe an beliebige Cursorpositionen möglich, was in Simon's Basic durch PRINT AT bewirkt wird.
Die Aufnahme von Cursorpositionierungen in einen String, z.B. A$=AT(10,20), ist im Gegensatz zu Simon's Basic nicht möglich; dafür können aber dem CHAR-Befehl die Cursorspalte und -zeile in Form von Variablen übermittelt werden, was auf die gleiche Wirkung hinausläuft.

DESIGN und MEM - Zeichensatz umdefinieren

In Basic 7.0 fehlen solche Befehle völlig.

CGOTO - GOTO zu errechneter Zeile

Dies ist nur mit der Routine »Label 128« (siehe 3.4.2) möglich.

DIR - Directory anzeigen

Dies wird durch DIRECTORY oder den gleichbedeutenden Befehl CATALOG auch auf dem C128 erreicht. Während DIR aber immer die Angabe des Filenamens (»$«) erfordert, kann DIRECTORY/CATALOG auch ohne Parameter stehen.

```
DIRECTORY           entspricht:   DIR"$"
DIRECTORY "*=PRG"   entspricht:   DIR"$:*=PRG"
```

DISK - Diskbefehl senden

Die wichtigsten Diskettenkommandos liegen beim C128 auf entsprechenden Basic-7.0-Befehlen:

```
COLLECT   DISK"V"
CONCAT    DISK"C:NEUFILE=ALTFILE1,ALTFILE2"
COPY      DISK"C:ZIELFILE=QUELLFILE"
HEADER    DISK"N:NAME,ID"
RENAME    DISK"R:NEUNAME=ALTNAME"
SCRATCH   DISK"S:LOESCHFILE"
```

Andere Kommandos müssen, wie in Basic 2.0 auch, über

```
OPEN 1,8,15,"Kommando als String":CLOSE 1
```

gesendet werden. Eine direkte Entsprechung des Simon's-Basic-Befehls DISK gibt es also nicht.

FETCH - Kontrollierte Eingaben

Außer INPUT bietet Basic 7.0 keine Möglichkeit, Eingaben zu holen. In 3.6.4 werden wir jedoch eine professionelle Eingaberoutine schreiben, die den FETCH-Befehl von Simon's Basic um Längen schlägt, aber auch als Ersatz für FETCH verwendbar ist.

INKEY - Funktionstastenabfrage

Da der C128 die Funktionstasten belegt, ist eine Abfrage nur möglich, wenn der C64-Zustand hergestellt wird (siehe 3.5.3).

LIN – aktuelle Cursorzeile ermitteln

Dies ist beim C128 nur mit PEEK möglich:

PEEK(235) ergibt die Cursorzeile
PEEK(236) ergibt die Cursorspalte

PAUSE – Pausenfunktion

In Basic 7.0 kann SLEEP verwendet werden. Soll zusätzlich eine Meldung gedruckt werden, muß ein eigenständiger PRINT- oder CHAR-Befehl stehen (PAUSE in Simon's Basic druckt auf Wunsch auch eine Meldung aus, was SLEEP nicht erledigt).

RESET – RESTORE auf errechnete Zeile

In Basic 7.0 kann RESTORE den DATA-Zeiger auch auf errechnete Zeilen positionieren (siehe 3.4.2). Beispiel:

Simon's Basic: RESET (C+50)*2
Basic 7.0: RESTORE (C+50)*2

– Befehle, die nicht im Simon's-Basic-Handbuch stehen

BCKGNDS – Hintergrundfarben festlegen und »Extended Color Mode«

Die Bildschirmfarben können über COLOR gesetzt werden; der »Extended Color Mode« ist eine Grafikbesonderheit, die von Basic 7.0 nicht unterstützt wird.

COLOUR – Rahmen- und Hintergrundfarbe setzen

In Basic 7.0 fällt dies unter COLOR. Beispiel:

Simon's Basic: COLOUR 0,1
Basic 7.0: COLOR 4,1:COLOR 0,2

Die Codes sind bei COLOR (Basic 7.0) um 1 höher als bei COLOUR (Simon's Basic).

DISABLE, RESUME und ON KEY – Tastaturabfrage

Befehle dieser Art gehören nicht zum Basic-7.0-Befehlsvorrat.

GRAPHICS – Konstante 53248 (VIC-Basisadresse)

Dieser Befehl ist durch die Zahl 53248 zu ersetzen. Beispiel:

Simon's Basic: POKE GRAPHICS+32,0
Basic 7.0: POKE 53248+32,0

NRM - Gegenstück zu MEM und BCKGNDS

Da für MEM und BCKGNDS in Basic 7.0 keine Äquivalente vorhanden sind, trifft dies folglich auch auf NRM zu.

SOUND - Konstante 54272 (SID-Basisadresse)

Dieser Befehl ist durch die Zahl 54272 zu ersetzen. Beispiel:

Simon's Basic: `POKE SID+24,0`
Basic 7.0: `POKE 54272+24,0`

3.6 Anwendungen zum Basic 7.0

Dieses Unterkapitel behandelt einige Anwendungen des Basic 7.0, deren Kenntnis Voraussetzung ist. Es werden dann

- Unterroutinen zur Weiterverwendung beschrieben
- Beispielprogramme zu Basic-7.0-Befehlen vorgestellt
- Programmiertechniken erklärt

Die Unterroutinen dürfen Sie gerne in eigenen Programmen weiterverwenden und diese nach Belieben verbreiten, solange Sie am Anfang jeder verwendeten Unterroutine einen Hinweis auf deren Herkunft in einer REM-Zeile anbringen (Titel, Autor, Verlag und Bestellnummer dieses Buches sowie die Seitenzahl).
Dies ist fairer Programmierstil und insofern kein Hindernis.

Hier ein kleiner Überblick über dieses Unterkapitel:

3.6.1 Routine für Balkengrafiken
Professionelle Geschäftsgrafiken für 80-Zeichen-Bildschirm mit bis zu 80 Balken und großer Flexibilität

3.6.2 Roulette mit hochauflösender Grafik und Sprites
Eines der beliebten Gesellschaftsspiele, programmiert mit Basic-7.0-Befehlen für hochauflösende Grafik, Sprites und Joystickabfrage und strukturierte Programmierung

3.6.3 Gerät verfügbar? - Prüfroutine aufrufen
Entwicklung eigener Prüfroutinen für den Zustand externer Geräte (Drucker/Floppy)

3.6.4 Professionelle Eingaberoutine
Eingaberoutine mit vielen Möglichkeiten (u.a. Abfangen von Fehleingaben) und Ausbaufähigkeit bis zur Eingabemaske

3.6.5 Komfortable Menüs einfach programmiert

3.6.6 Farbeinstellung über Menü
Lösen Sie das Farbwahl-Problem ein für allemal

3.6.7 Windows - Fenster zum Bedienungskomfort
Programmierung von »echten« Windows mit dem C128

3.6.1 Routine für Balkengrafiken auf 80-Zeichen-Bildschirm

Eine Anwendung des Computers ist der geschäftliche Bereich, für den der C128 durch seine 80-Zeichen-Fähigkeit und die für den CP/M-Modus erhältliche Software prädestiniert ist.
Für demoskopische Programme, Wirtschaftsbilanzen und ähnliches aus dem statistischen Bereich werden zur Veranschaulichung von Zahlenwerten Balkengrafiken verwendet, die die trockenen Zahlen in übersichtliche Grafiken umwandeln. Für jeden Zahlenwert steht ein Balken, der in seiner Höhe proportional zum numerischen Wert, den er repräsentiert, ist.
In der Regel programmiert man solche Balkengrafiken unter Einsatz der hochauflösenden Grafik, aber diese Routine zeigt, daß es auch anders geht (laden Sie dieses Programm von Diskette, schalten Sie Ihren 80-Zeichen-Monitor ein und starten Sie das Programm):
Beschreibung: Balkengrafiken auf dem 80-Zeichen-Bildschirm
Filename: »balkengrafik«

```
100 REM ***********************************
110 REM *                                 *
120 REM *  B A L K E N D I A G R A M M    *
130 REM *                                 *
140 REM *      FUER 80-ZEICHEN-MODUS      *
150 REM *                                 *
160 REM ***********************************
170 :
180 :
190 A =14:     REM >>> FARBWIEDERHOLZYKL.(14) <<<
200 B =22:     REM >>> ANZAHL DER BALKEN (70) <<<
210 C = 2:     REM >>> BALKENBREITE      1-74 <<<
220 D =20:     REM >>> BALKENHOEHE       (22) <<<
230 E = 3:     REM >>> RASTERABSTAND     1-74 <<<
240 S = 4:     REM >>> LINKER GRAFIKRAND (50) <<<
250 :
260 REM BENUTZTE VARIABLEN : WE() / LW() / IV() / A-V
270 REM                      WE() = WERTE, DIE DARGESTELLT WERDEN
280 :
290 DATA 164,175,185,162,184,183,163:REM ASCII-CODES DER GRAFIKZEICHEN, DIE FUER
 BALKENDARSTELLUNGEN BENOETIGT WERDEN
300 :
310 DATA 28,150,30,153,31,154,155,5,129,156,151,159,149,158:REM ASCII-CODES DER
STEUERZEICHEN FUER AENDERUNG DER ZEICHENFARBE (= BALKENFARBE)
320 REM IN REIHENFOLGE: ROT,HELLROT,GRUEN,HELLGRUEN,BLAU,HELLBLAU,HELLGRAU,WEISS
,DUNKEL-LILA,LILA,DUNKELGRAU,TUERKIS,BRAUN,GELB
330 :
340 DIM WE(B),LW(B),IV(B):REM ARRAYS DIMENSIONIEREN
350 :
360 FORT=1TOB:WE(T)=INT(RND(1)*345)+5:NEXT :REM WERTE NACH ZUFALL BERECHNEN LASS
EN
370 :
380 GRAPHIC 5,1:BANK 15:FAST:SCNCLR:PRINTTAB(20)"BALKENDIAGRAMM WIRD ERRECHNET"
390 G=0:FOR H=1 TO B  :REM WERTE IN BALKENLAENGE UMSETZEN (MAXIMALE BALKENHOEHE
- IN VARIABLE 'D' ENTHALTEN - BERUECKSICHTIGEN)
400 I=0:LW(1)=146
410 IFWE(H)/(10↑I)>DTHENI=I+.3  :IFG<ITHEN G=I:REM  +.3 = AUFLOESUNG (!)
420 IFWE(H)/(10↑I)>DTHEN410
430 NEXTH
440 :
450 FORH=1TOB:WE(H)=(WE(H)/(10↑G)):J=INT(WE(H)):REM TEILER
460 FORK=0TO6:IFWE(H)>J+(.142*K)THENREADL:IV(H)=146:IFK>3THENIV(H)=18
```

```
470 NEXTK:RESTORE 290:LW(H)=L:L=0:NEXTH :REM LETZTES GRAFIKZEICHEN HOLEN
480 FORH=1TOB:FORM=0TO23
490 IFINT(WE(H))=MTHENWE(H)=23-M:GOTO510 :REM REZIPROKER WERT DER ZAHLEN
500 NEXTM
510 NEXTH
520 :
530 SCNCLR :REM NUMERIERUNG DER Y-ACHSE
540 CHAR,2,22-D:PRINTTAB(S)CHR$(5);"> INDEXMULTIPLIKATOR ="INT((10↑G)*2+.5)/2;"<
"CHR$(30):REM INDEXMULTIPLIKATOR ANZEIGEN
550 FORH=DTO1STEP-1:IFH<10THENPRINTTAB(S-4);CHR$(5);HCHR$(30);"┌":ELSEPRINTTAB(S
-4);CHR$(157);CHR$(5);H;CHR$(30);"┌"
560 NEXTH
570 :
580 CHAR,S-1,23
590 FORH=1TO (B*E)+(C-E)+2:PRINTCHR$(30);"‾";:NEXTH:PRINT :REM NUMERIERUNG DER X
-ACHSE
600 PRINTTAB(S+1):FORT=1TO(B*E)+S:IFE>1THEN FORU=1TO10:V=2:IFU=10THENU=0
610 IFE<2THEN650
620 DO WHILEV<EANDT>1:PRINT" ";:V=V+1:LOOP:T=T+E:PRINTCHR$(5);CHR$(157);U;:IFT>B
*E THEN680
630 NEXTU,T :REM EINTEILUNG BEI BALKENABSTAND <1
640 :
650 I=0:FORH=1TOE*B:I=I+1:PRINTCHR$(5);MID$("1234567890",I,1);:IFI=10THENI=0
660 NEXTH
670 :
680 N=0:O=0:RESTORE310:FORH=1TO(B*E)STEPE:READR:O=O+1:IFO=ATHENO=0:RESTORE 310 :
REM BALKENFARBE LESEN
690 N=N+1
700 :
710 FORP =22TOWE(N)STEP-1
720 CHAR,H+S,P
730 IFWE(N)>22THEN740:ELSEPRINTCHR$(R);CHR$(18);:FORF=1TOC:PRINTCHR$(32);:NEXTF
:REM GRUNDBALKEN DARSTELLEN
740 NEXTP:IFWE(N)=23THENQ=P+1:ELSEQ=P
750 :
760 CHAR,H+S,Q:PRINTCHR$(R);CHR$(IV(N));:FORF=1TOC:PRINTCHR$(LW(N));:NEXTF,H : R
EM GRAFIKZEICHEN AUFSETZEN
770 GETKEY W$:SCNCLR:FORH=1TOB:LW(H)=0:WE(H)=0:IV(H)=0:NEXTH :REM VARIABLE LOESC
HEN
780 REM "RETURN" BEI VERWENDUNG ALS UNTERROUTINE ANFUEGEN
```

Da Sie nun einen Eindruck von der Wirkung dieser Routine bekommen haben, wollen wir dieses Programm auch besprechen.

Vorzüge der Routine

- automatische Wertanpassung
- variable Höhe und Breite der Grafik
- variable Positionierung der Grafik
- variable Balkenfarbe
- frei definierbarer Balkenabstand
- hohe Verarbeitungsgeschwindigkeit
- geringer Speicherplatzbedarf (ca. 3K), gemessen an der Leistung
- Anzeige des Indexmultiplikators

Die Rechengeschwindigkeit rührt daher, daß anstatt der etwas schwerfälligen hochauflösenden oder Multicolor-Grafik die Commodore-Grafikzeichen ausgiebig eingesetzt werden.

Anwendung der Routine

Wenn diese Routine zu einem Unterprogramm umfunktioniert werden soll, ist nur in Zeile 780 der RETURN-Befehl anstelle des REM einzusetzen. Ein Umnumerieren durch RENUMBER bereitet keine Schwierigkeiten und ist wahrschweinlich sinnvoll, um das Programm in einen höheren Zeilenbereich (ab 50000) zu legen.
Die Berechnung zufälliger Werte zum Zweck der Demonstration, die in Zeile 360 steht, sollte entfernt werden.
Bei der Parameterübergabe haben Sie folgende zwei Alternativen:

1. Parameter in Unterroutine einbauen

Bei nur einmaligem Aufruf der Routine kann man die Parameter, die noch besprochen werden, in den Zeilen 190-250 anstelle der dort stehenden Variablendefinitionswerte einsetzen. Auf jeden Fall müssen vor Aufruf der Routine im Feld WE() die darzustellenden Werte enthalten sein, wobei der Index 0, also das Element WE(0), nicht verwendbar ist, da das Programm erst mit Index 1, also dem Element WE(1), bei der Auswertung beginnt. Die Anzahl der Werte geht aus der Anzahl der Balken (Variable B, Zeile 200) hervor.
Die Dimensionierung des Feldes WE() in Zeile 340 muß entfernt werden (aber vorher im Hauptprogramm erfolgt sein):

```
340 DIM LW(B),IV(B)
```

2. Parameter aus Hauptprogramm übergeben

Falls die Unterroutine mehrfach mit verschiedenen Parametern aufgerufen werden soll, übermittelt am besten das Hauptprogramm in bestimmten Variablen die benötigten Werte. Dazu sind einige Zeilen zu entfernen, da sonst die (für Demonstrationszwecke vorgesehene) Belegung der Variablen erfolgt. Hier die DELETE-Anweisungen:

```
DELETE 190-240
DELETE 360
```

In Zeile 340 muß die Dimensionierung des Feldes WE() aufgehoben werden, die vorher im Hauptprogramm zu erfolgen hat:

```
340 DIM LW(B),IV(B)
```

Bedeutung der Parameter

Unabhängig davon, ob Sie sich für die 1. oder 2. Alternative entschieden haben, die Bedeutung der Parameter ist immer gleich:

– *Array WE(): numerische Werte zur Darstellung*

Die Zahlen, die als Balken dargestellt werden sollen, stehen im Array WE(). Der Index 0 wird nicht verwendet, der erste Wert steht also in WE(1).
Nach dem Zeichnen der Grafik ist dieses Array gelöscht (mit 0 belegt); diese Werte müssen also vor dem Aufruf der Routine in ein anderes Array gerettet werden, falls eine Weiterverwendung geplant ist.

- *Variable A: Farbwiederholzyklus*

Daraus geht hervor, nach wieviel verschiedenen Farben die Farbreihenfolge wieder von vorne beginnt (Beispiel: A=6, nach sechs verschiedenen Farben werden diese wieder in der gleichen Reihenfolge wiederholt). Dieser Wert ist minimal 1, maximal 16.

- *Variable B: Anzahl der Balken*

In B wird die Anzahl der benötigten Balken abgelegt. Diese Zahl ist abhängig von der Position der Grafik: Je weiter rechts sie steht, desto geringer ist die Zahl der möglichen Balken (maximal 70).
Aus B geht auch hervor, wie groß das Feld WE() bzw. der daraus erwünschte Teilausschnitt zur Bildschirmdarstellung ist. Deshalb ist B gleichzeitig die Anzahl der Werte (1 Balken repräsentiert genau 1 Wert).

- *Variable C: Balkenbreite*

Auch dieser Wert ist abhängig von der Anzahl der Balken und der Bildschirm-Ausgabeposition. Die Breite darf maximal 74 sein.

- *Variable D: Balkenhöhe*

Die Höhe der Balken liegt im Bereich 1-22 und ist nicht von anderen Werten wie der Anzahl der Balken oder der Bildschirm-Ausgabeposition abhängig.

- *Variable E: Rasterabstand*

Dieser Wert gibt die Zahl der freien Punkte (Leerzeichen) zwischen den Balken an, ist von der Anzahl der Balken und der Bildschirm-Ausgabeposition abhängig und liegt im Bereich 3-74. Die Werte 1 und 2 sollten nur in Ausnahmefällen verwendet werden, 0 ergibt ein lustiges Farbenspiel (allerdings ansonsten kein brauchbares Resultat) und negative Zahlen führen zu Fehlfunktionen.

- *Variable S: Anfangsposition der Grafik vom linken Bildschirmrand*

Die Angabe ist eine Spaltenzahl (mindestens 4), deren Wertebereich von Anzahl der Balken, Balkenbreite und Rasterabstand abhängt.

- *Zulässigkeit der Parameter*

Es ist zu beachten, daß

```
Balkenanzahl * Balkenbreite + Rasterabstand + Abstand links
```

den Wert 75 nicht überschreiten darf! Damit dies vom Programm geprüft wird, ist folgende Zeile zu ergänzen:

```
280 IF B*C+E+S>75 THEN PRINT"UNZULAESSIGE PARAMETER!":STOP
```

Die Wirkung bestimmter Parameter kann aber auch mit dem Programm »c/balken-demo« überprüft werden, das aus Geschwindigkeitsgründen compiliert (von Basic in Maschinensprache mit Hilfe eines sogenannten »Compiler«-Programms übersetzt) ist und infolgedessen nicht gelistet werden kann.
Dieses Programm bietet zunächst die Wahl zwischen eigener Eingabe der Werte WE() oder zufälliger Ermittlung durch den Computer (<RETURN>). Dann können Sie nacheinander die Parameter eingeben, wobei der maximale Wert immer angezeigt wird. Dabei wird die genannte Formel zur Berechnung zulässiger Parameter eingesetzt, um Ihnen jeweils den erlaubten Wert anzuzeigen.

Beschreibung der Routine

Zur Anwendung des Balkengrafik-Unterprogramms ist das Durchlesen der nun folgenden Dokumentation nicht erforderlich, zum Verständnis allerdings schon.

- *Zeilen 190-240: Parameterdefinitionen*

Diese Parameterdefinitionen sind bei eigenen Parametern zu entfernen oder zumindest durch andere Variablenzuweisungswerte zu ersetzen.

- *Zeile 290: ASCII-Codes der Grafikzeichen*

In Zeile 290 stehen die ASCII-Codes der Grafikzeichen, die zum Aufbau der Balken benötigt werden. Die Bedeutung können Sie der »Commodore-128-Codetabelle« im Anhang des C128-Handbuches entnehmen, die dort auf Seite A-7 beginnt.

- *Zeile 310: ASCII-Codes der Cursorfarben*

Da jede Schriftfarbe außer über »COLOR 5,Farbcode« auch durch »PRINT CHR$(x)« erreicht werden kann, wobei für »x« nicht einfach der Farbcode, sondern ein spezieller ASCII-Code für ein Steuerzeichen eingesetzt werden muß, stehen diese Codes in der DATA-Zeile 310; übrigens genau in der Reihenfolge, die bei den Balkenfarben auch eingehalten wird.
Die Bedeutung eines einzelnen ASCII-Codes können Sie der darauffolgenden Zeile 320 entnehmen.

- *Zeile 340: Dimensionierung der Arrays*

Dort ist »WE(B),« zu entfernen, wenn eigene Werte in WE() übergeben werden sollen. Andernfalls kann dies zu einem »?REDIM'D ARRAY ERROR« führen.

- *Zeile 360: Zufällige Berechnung der Werte*

Diese Berechnung muß gelöscht werden, wenn eigene Werte in WE() übergeben werden, da ansonsten die ursprünglichen Werte verlorengehen und demzufolge eine falsche Grafik angezeigt wird.

- *Zeile 380: Initialisierung*

In Zeile 380 wird auf den 80-Zeichen-Bildschirm geschaltet (GRAPHIC 5), Bank 15 eingestellt, der FAST-Modus angewählt und der Bildschirm gelöscht, bis schließlich die Meldung »Balkendiagramm wird errechnet« ausgegeben wird.
Diese Zeile kann noch optimiert werden, da Bank 15 nicht eingestellt werden muß und die Angabe von »,1« hinter GRAPHIC 5 bereits das Löschen des Bildschirms bewirkt:

```
380 GRAPHIC 5,1:FAST:PRINTTAB(20)"BALKENDIAGRAMM WIRD ERRECHNET"
```

- *Zeilen 390-430: Umsetzung der Werte in Balkenlängen*

Nach dieser Schleife steht in der Variablen G ein Wert, aus dem später der Indexmultiplikator hervorgeht. Der Indexmultiplikator wird zur Anpassung der unterschiedlichen Werte auf das Bildschirmformat benötigt und auch ausgegeben.

- *Zeilen 450-510: Array WE() umrechnen*

Diese Schleife ändert die Werte im Array WE(), um auf leichtere Weise die Balkenlänge entnehmen zu können.

- *Zeilen 530-560: Numerierung der Y-Achse*

Die Aufgabe dieser Schleife ist die Anzeige des Indexmultiplikators, der in 390-430 vorbereitet wird, und die Numerierung der Y-Achse entsprechend den Werten, die später angezeigt werden.

- *Zeilen 580-660: Numerierung der X-Achse*

Auch die X-Achse wird numeriert.

- *Zeilen 680-760: Anzeige der berechneten Grafik*

Die Grafik wurde hauptsächlich in den Zeilen 390-510 berechnet, die endgültige Ausgabe der Balken erfolgt aber erst an dieser Stelle.

- *Zeile 770: Warten auf Tastendruck und Löschen einiger Werte*

Nach GETKEY W$ (Warten auf einen Tastendruck) werden einige Hilfswerte des Programms, die in Arrays enthalten sind, gelöscht. Dies betrifft auch die Werte in WE(), die also vor dem Aufruf der Routine in ein anderes Array gerettet werden müssen, wie schon bei der Beschreibung der Parameter erklärt wurde.

Umschreiben der Routine für Betrieb mit dem 40-Zeichen-Bildschirm

Ist ein Programm für den Betrieb mit zwei Bildschirmen (40 und 80 Zeichen pro Zeile) konzipiert, können die Werte zumindest teilweise (wenn es sich um eine große Anzahl handelt) auf

dem 40er-Bildschirm angezeigt werden, während die Grafik auf dem 80-Zeichen-Bildschirm steht.
Wenn die entsprechenden Programmteile geändert werden (Initialisierung in Zeile 380) und das Programm auf die verringerte Bildschirmbreite eingestellt wird, kann die Routine auch am 40-Zeichen-Bildschirm laufen. Ich habe mich jedoch deshalb für den 80er-Modus entschieden, weil dieser mehr Platz für solche aufwendigen Bildschirmgrafiken bietet und für Geschäftsanwendungen wie Bilanzen vorgesehen ist. Für wenige Werte reicht aber unter Umständen auch der 40er-Modus aus.

3.6.2 Roulette mit hochauflösender Grafik

Nachdem in 3.6.1 die Programmierer von Software für den geschäftlichen Bereich mit einer Routine für Balkengrafiken versorgt wurden, sind in diesem Unterkapitel diejenigen an der Reihe, die sich auch für weniger »ernste« Anwendungen begeistern können.
Das beliebte Spiel »Roulette« soll die Basic-7.0-Kommandos für folgende Bereiche verdeutlichen und nebenbei auch noch Spaß machen:

- Strukturierte Programmierung
- Grafik
- Ein-/Ausgabe
- Fehlerbehandlung
- Joystick-Abfrage

Beschreibung: Roulette für den 40-Zeichen-Modus
Filename: »roulette«

```
0 REM ******************************
1 REM *                            *
2 REM *  R O U L E T T E   1 2 8   *
3 REM *                            *
4 REM ******************************
5 :
10 IF RWINDOW(2)=80 THEN PRINT"PROGRAMM LAEUFT GERADE AUF 40-ZEICHEN-MONITOR !":
GRAPHIC0,1
11 FAST:REM FUER BILDSCHIRMAUFBAU DOPPELTE GESCHWINDIGKEIT EINSCHALTEN
12 TRAP 7000:REM EIGENE FEHLERBEHANDLUNGSROUTINE
15 SCNCLR:COLOR 0,6:GRAPHIC 1,1
16 POKE0,PEEK(0) AND 63:REM AMERIKANISCHE TASTATUR
20 DIMZ(10),S$(10,60),S(10),N$(10),K(6),A$(6,20),A(6),NS$(6),X$(23,6)
30 SP=1:ZU=RND(-TI):REM ZUFALLSGENERATOR INITIALISIEREN
100 DATA37,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
102 DATA25,26,27,28,29,30,31,32,33,34,35,36,36,PLEIN
105 DATA60,0001,0002,0003,0102,0104,0203,0205,0306,0405,0407,0506,0508
106 DATA0609,0708,0710,0809,0811,0912,1011,1013,1112,1114,1215,1314,1316
107 DATA1415,1417,1518,1617,1619,1718,1720,1821,1920,1922,2021,2023,2124
108 DATA2223,2225,2324,2326,2427,2526,2528,2627,2629,2730,2829,2831,2930
109 DATA2932,3033,3132,3134,3233,3235,3336,3435,3536,18,CHEVAL
110 DATA14,010203,040506,070809,101112,131415,161718,192021,222324
111 DATA252627,282930,313233,343536,000102,000203,12,TRANSVERSALE PLEIN
115 DATA23,00010203,01020405,02030506,04050708,05060809,07081011
```

```
116 DATA08091112,10111314,11121415,13141617,14151718,16171920,17182021
117 DATA19202223,20212324,22232526,23242627,25262829,26272930
118 DATA28293132,29303233,31323435,32333536,9,CARRE
120 DATA11,010203040506,040506070809,070809101112,101112131415,131415161718
121 DATA161718192021,192021222324,222324252627,252627282930,282930313233
122 DATA313233343536,6,TRANSVERSALE SIMPLE
125 DATA3,010203040506070809101112,131415161718192021222324
126 DATA252627282930313233343536,3,DOUZE
130 DATA3,010407101316192225283134,020508111417202326293235
131 DATA030609121518212427303336,3,KOLONNE
135 DATA2,010203040506070809101112131415161718
136 DATA192021222324252627282930313233343536,2,MANQUE/PASSE
140 DATA2,010305070911131517192123252729313335
141 DATA020406081012141618202224262830323436,2,IMPAIR/PAIR
145 DATA2,020406081011131517202224262829313335
146 DATA010305070912141618192123252730323436,2,SCHWARZ/ROT
150 DATA401,201,313,202,314,203,401,301,102,204,103,206,104,301
152 DATA501,205,402,207,403,208,501,302,105,209,106,211,107,302
154 DATA502,210,404,212,405,213,502,303,108,214,109,216,110,303
156 DATA503,215,406,217,407,218,503,304,111,219,112,221,113,304
158 DATA504,220,408,222,409,223,504,305,114,224,115,226,116,305
160 DATA505,225,410,227,411,228,505,306,117,229,118,231,119,306
162 DATA506,230,412,232,413,233,506,307,120,234,121,236,122,307
164 DATA507,235,414,237,415,238,507,308,123,239,124,241,125,308
166 DATA508,240,416,242,417,243,508,309,126,244,127,246,128,309
168 DATA509,245,418,247,419,248,509,310,129,249,130,251,131,310
170 DATA510,250,420,252,421,253,510,311,132,254,133,256,134,311
172 DATA511,255,422,257,423,258,511,312,135,259,136,260,137,312
199 :
200 REM DATEN EINLESEN
201 :
205 FORI=1TO10
210 READ Z(I)
215 FORJ=1TOZ(I)
220 READ S$(I,J)
225 NEXTJ
230 READ S(I),N$(I)
235 NEXTI
240 :
245 FORI=0TO23
250 FORJ=0TO6
255 READ X$(I,J)
256 IF LEN(X$(I,J))<>3 THEN SLOW:PRINT"FEHLER IN DATAS AB ZEILE 150!!":END
260 NEXTJ,I
299 :
300 REM GRAFIK AUFBAUEN
301 :
310 COLOR 0,6:COLOR 1,1:COLOR 4,14:REM BILDSCHIRMFARBEN EINSTELLEN
315 GRAPHIC 1:REM HOCHAUFLOESENDE GRAFIK EINSCHALTEN
330 DATA3,1,3,1,3,1,3,1,3,1,1,3,1,3,1,3,1,3,3,1,3,1,3,1,3,1,3,1,1,3,1,3,1,3,1,3
331 H=1
335 FORI=11TO22
340 FORJ=17TO21STEP2
345 READA
350 A$=STR$(H):IFH>9THENA$=RIGHT$(A$,2)
355 COLOR1,A
360 CHAR,J,I,A$,1
365 H=H+1
370 NEXTJ
375 NEXTI
380 COLOR1,1:CHAR,20,10,"0"
382 CHAR,12,13,"PASSE"
384 CHAR,23,13,"MANQUE"
```

```
386 CHAR,12,16,"PAIR"
388 CHAR,23,16,"IMPAIR"
395 BOX,137,80,184,87
400 DRAW,137,79TO88,88TO88,188TO137,192TO184,192TO232,189TO232,88TO184,79
405 DRAW,136,119TO89,120
410 DRAW,136,151TO89,150
415 DRAW,89,180TO137,184TO184,184TO231,180
420 DRAW,184,119TO231,120
425 DRAW,184,151TO231,150
430 DATA 104,182,189,120,184,191,136,185,191,152,185,191,168,185,191,184,185,191
431 DATA 200,184,191,216,182,189
435 FORI=1TO8
440 READ X,Y1,Y2
445 DRAW,X,Y1TOX,Y2
450 NEXTI
455 DRAW,104,163TO109,153TO110,153TO115,163TO115,164TO110,174TO109,174
460 DRAW,109,174TO104,164
465 PAINT,110,165
466 COLOR1,3
470 DRAW,204,163TO209,153TO210,153TO215,163TO215,164TO210,174TO209,174
475 DRAW,209,174TO204,164
480 PAINT,210,165
485 COLOR1,1
490 DATA0,0,4,1,0,4,2,0,4,0,1,4,0,2,4,2,1,4,2,2,4,1,2,4,0,3,4,0,4,4
492 DATA16,3,0,16,4,0,16,5,0,16,6,0,17,3,0,18,4,0,19,3,0,20,3,0,20,4,0,20,5,0
494 DATA20,6,0
496 DATA32,4,-3,33,4,-3,34,5,-3,34,6,-3,33,7,-3,32,5,-3,32,6,-3,32,7,-3
500 FORI=1TO29
505 READX,Y,Y1
510 DRAW,94+X,182+Y
515 DRAW,193+X,182+Y+Y1
520 NEXTI
530 FORW=0TO360 STEP(360/15)
531 CIRCLE,160,40,30,,W,W+1
532 NEXTW
533 CIRCLE,160,40,36:CIRCLE,160,40,39
534 PAINT,197,40
535 COLOR1,8
540 CIRCLE,160,40,27
545 CIRCLE,160,40,15
550 COLOR1,16
555 CHAR,19,4,"\/"
556 CHAR,19,5,"/\"
560 COLOR1,8
565 FORW=0 TO·360 STEP(360/37)
570 CIRCLE,160,40,15,,W,W+1
572 X=RDOT(0):Y=RDOT(1)
574 V=(W*π)/180
576 X1=160+27*SIN(V):Y1=40-27*COS(V)
578 DRAW,X,YTOX1,Y1
580 NEXTW
585 COLOR1,10
590 DRAW,75,0TO75,199
592 DRAW,78,0TO78,199
594 PAINT,77,1
595 DRAW,242,0TO242,199
597 DRAW,245,0TO245,199
599 PAINT,244,1
600 DRAW,78,196 TO 242,196
602 PAINT,80,198
605 COLOR1,1
610 CIRCLE,6,6,3:PAINT,6,6
612 SSHAPE A$,0,0,23,20
```

```
615 SPRSAV A$,1
620 CIRCLE0,6,6,3:PAINT0,6,6
625 SPRITE1,0,15,0,0,0,0
630 COLOR1,1
635 DRAW,0,0TO1,0TO1,1TO0,1
640 SSHAPE A$,0,0,23,20
645 SPRSAV A$,2
650 DRAW0,0,0TO1,0TO1,1TO0,1
655 SPRITE2,0,1,0,0,0,0
700 POKE0,PEEK(0)OR 64:POKE1,PEEK(1)AND63:REM DEUTSCHE TASTATUR
998 SLOW:REM MACHT FAST-BEFEHL AUS ZEILE 11 RUECKGAENGIG, DAMIT DIE GRAFIK SICHT
BAR WIRD
999 :
1000 REM EINGABE DER SPIELERNAMEN
1001 :
1002 COLOR 1,2:REM ZEICHENFARBE IN GRAFIK EINSTELLEN
1003 GOSUB 6000:REM RECHTEN GRAFIKBEREICH LOESCHEN
1005 CHAR,31,2,"NAME DES":CHAR,31,3,STR$(SP)+".":CHAR,31,4,"SPIELERS?"
1010 P2=6:H1$="A":H2$="┘":GOSUB 6200:REM EINGABE EINES STRINGS HOLEN
1015 NS$(SP)=B$:REM SPIELERNAMEN WERDEN IM ARRAY N$() GESPEICHERT
1020 CHAR,31,10,"KAPITAL":CHAR,31,11,"DES":CHAR,31,12,STR$(SP)+"."
1024 CHAR,31,13,"SPIELERS":CHAR,31,14,"IN DM :"
1025 P2=16:H1$="0":H2$="9":GOSUB6200:REM EINGABE DES KAPITALS
1045 K(SP)= VAL(B$):REM SPIELERKAPITAL WIRD IM ARRAY K() GESPEICHERT
1050 CHAR,31,20,"EINGABEN":CHAR,31,21," OK ?"
1055 GETKEY A$
1060 IF A$="N" THEN 1000
1065 IF A$="J" THEN 1075
1070 GOTO 1055
1075 CHAR,0,SP*4,NS$(SP)+":":CHAR,0,SP*4+1,STR$(K(SP)):REM SPIELERNAME UND KAPIT
AL IN DIE GRAFIK SCHREIBEN
1100 GOSUB6000:REM RECHTE SEITE DER GRAFIK LOESCHEN
1103 IF SP=5 THEN 2000:REM MAXIMAL 5 SPIELER MOEGLICH
1105 CHAR,31,5,"STEIGT":CHAR,31,6,"NOCH":CHAR,31,7,"EIN"
1110 CHAR,31,8,"SPIELER":CHAR,31,9,"EIN ?"
1115 GETKEY A$
1120 IF A$="J" THEN SP=SP+1:GOTO 1000
1125 IF A$ <> "N" THEN 1115
1999 :
2000 REM SPIELEINGABE
2001 :
2002 GOSUB 6000:REM RECHTE SEITE DER GRAFIK LOESCHEN
2005 FOR S= 1 TO SP
2006 SE=1:GRAPHIC1:GOSUB6000
2008 FOR I= 1 TO SP
2009 CHAR,0,I*4+1,"            ":CHAR,0,I*4+1,STR$(K(I)):NEXT
2010 CHAR,32,2,NS$(S)
2011 CHAR,32,5,"M",1:CHAR,33,5,"EN┘"
2012 CHAR,33,7,"ODER"
2014 CHAR,32,9,"T",1:CHAR,33,9,"ABLEAU"
2015 CHAR,35,11,"?"
2020 GETKEY A$
2022 IF A$="M" THEN GRAPHIC 0:GOTO 2500
2024 IF A$<>"T"THEN 2020
2030 SPRITE 1,1
2035 GOSUB6000
2087 AN=0:GOSUB6000
2100 MOVSPR 1,115,229
2110 J=JOY(2)
2111 X=RSPPOS(1,0):Y=RSPPOS(1,1)
2113 IFJ=128THEN2300
2114 IFJ=0ANDAN=1THEN2110
2115 IFJ=0THENGOSUB6000:GOSUB7500:GOSUB8000:AN=1:GOTO2110
```

```
2118 AN=0
2120 IF J=1 THEN BEGIN
2121 :IFX>154ANDX<204ANDY>131ANDY<224THENMOVSPR1,+0,-4:GOTO2110
2122 :IFX=120ANDY=185THENMOVSPR1,+14,-46:GOTO2110
2123 :IFX=115ANDY=215THENMOVSPR1,+5,-30:GOTO2110
2124 :IF(X=115ORX=130ORX=145)AND(Y=229ORY=230ORY=231)THENMOVSPR1,115,215:GOTO211
0
2126 :IFY=232THENMOVSPR1,+1,-9:GOTO2110
2128 :IFX=214ANDY=218THENMOVSPR1,+18,-31:GOTO2110
2130 :IFX=232ANDY=187THENMOVSPR1,-14,-50:GOTO2110
2132 :IFY=131THENMOVSPR1,175,127:GOTO2110
2134 BEND
2140 IFJ=3THEN BEGIN
2142 :IFX>154ANDX<203ANDY>130ANDY<224THENMOVSPR1,+8,+0:GOTO2110
2144 :IFX=134ANDY=139THENMOVSPR1,+21,+12:GOTO2110
2146 :IFX=120ANDY=185THENMOVSPR1,+35,-10:GOTO2110
2148 :IFX=115ANDY=215THENMOVSPR1,+40,+0:GOTO2110
2150 :IF(X=115ORX=130)AND(Y=229ORY=230)THEN MOVSPR1,+15,+1:GOTO2110
2152 :IFX=145ANDY=231THENMOVSPR1,+17,+1:GOTO2110
2154 :IF(X=162ORX=178)ANDY=232THENMOVSPR1,+16,+0:GOTO2110
2156 :IFX=203ANDY>130ANDY<160THENMOVSPR1,218,137:GOTO2110
2158 :IFX=203ANDY>161ANDY<192THENMOVSPR1,232,187:GOTO2110
2160 :IFX=203ANDY>195ANDY<223THENMOVSPR1,214,218:GOTO2110
2165 BEND
2170 IFJ=5THEN BEGIN
2172 :IFX>154ANDX<204ANDY>130ANDY<223THENMOVSPR1,+0,+4:GOTO2110
2174 :IFX=175ANDY=127THENMOVSPR1,179,131:GOTO2110
2176 :IFX=134ANDY=139THENMOVSPR1,120,185:GOTO2110
2178 :IFX=120ANDY=185THENMOVSPR1,115,215:GOTO2110
2180 :IFX=115ANDY=215THENMOVSPR1,130,230:GOTO2110
2182 :IFX=232ANDY=187THENMOVSPR1,214,218:GOTO2110
2184 :IFX=218ANDY=137THENMOVSPR1,232,187:GOTO2110
2186 :IFY=223AND(X=163ORX=179ORX=195)THENMOVSPR1,-1,+9:GOTO2110
2190 BEND
2200 IF J=7THEN BEGIN
2202 :IFX>155ANDX<204ANDY>130ANDY<224THENMOVSPR1,-8,+0:GOTO2110
2204 :IFX=218ANDY=137THENMOVSPR1,203,151:GOTO2110
2206 :IFX=232ANDY=187THENMOVSPR1,203,175:GOTO2110
2208 :IFX=214ANDY=218THENMOVSPR1,203,207:GOTO2110
2210 :IF(X=130ORX=145)AND(Y=230ORY=231)THEN MOVSPR1,-15,-1:GOTO2110
2212 :IF(X=178ORX=194)ANDY=232THENMOVSPR1,-16,+0:GOTO2110
2214 :IFX=155ANDY>130ANDY<160THENMOVSPR1,134,139:GOTO2110
2216 :IFX=155ANDY>161ANDY<192THENMOVSPR1,120,185:GOTO2110
2218 :IFX=155ANDY>195ANDY<223THENMOVSPR1,115,215:GOTO2110
2220 :IFX=162ANDY=232THENMOVSPR1,145,231:GOTO2110
2222 BEND
2230 GOTO2110
2300 REM BILDEN DES STRINGS
2305 A$(S,SE)=E$+F$+"D"
2315 GOSUB6000
2320 CHAR,31,5,"WIEVIEL":CHAR,31,6,"DM WOLLEN"
2322 CHAR,31,7,"SIE":CHAR,31,8,"SETZEN ?"
2325 P2=10:H1$="0":H2$="9":GOSUB6200
2330 IFVAL(B$)>K(S)ORVAL(B$)<=0THEN2315
2335 K(S)=K(S)-VAL(B$)
2340 CHAR,0,S*4+1,"             ":CHAR,0,S*4+1,STR$(K(S))
2345 A$(S,SE)=A$(S,SE)+" "+B$
2346 A(S)=SE
2347 IFSE=20ORK(S)=0THEN2380
2350 GOSUB6000
2355 CHAR,31,1,"WOLLEN":CHAR,31,2,"SIE ,"
2360 CHAR,31,3,NS$(S):CHAR,31,4,", AUF":CHAR,31,5,"NOCH MEHR"
2365 CHAR,31,6,"SETZEN ?"
```

```
2370 GETKEYA$
2375 IFA$="J"THEN SE=SE+1:GOTO2087
2376 IFA$<>"N"THEN2370
2380 NEXTS
2385 GOTO3000
2500 SCNCLR
2505 SPRITE 1,0
2515 PRINTNS$(S)":":PRINT
2520 FORI=1TO10
2525 PRINTI,N$(I)
2530 NEXTI
2535 INPUT"IHRE WAHL (1,2,3...)";A$
2540 IFVAL(A$)<1ORVAL(A$)>10THEN2535
2545 IFVAL(A$)<10THENA$="0"+A$
2550 SCNCLR
2555 FORI=1TOZ(VAL(A$))
2560 PRINTCHR$(18);I;CHR$(146)" ";
2565 FORJ=1TO(LEN(S$(VAL(A$),I))-1)STEP2
2570 PRINTMID$(S$(VAL(A$),I),J,2)"/";
2575 NEXTJ
2580 PRINTCHR$(157);" "
2585 NEXTI
2590 INPUT"IHRE WAHL";B$
2595 IF VAL(B$)<1ORVAL(B$)>Z(VAL(A$))THEN2590
2600 INPUT"WIEVIEL DM";DM
2605 IF DM>K(S)ORDM<=0THEN2600
2610 K(S)=K(S)-DM
2615 A$(S,SE)=A$+S$(VAL(A$),VAL(B$))+"D"+STR$(DM)
2617 IFSE=20ORK(S)=0THEN2640
2620 PRINT"WILL SPIELER "S" AUF NOCH MEHR SETZEN?"
2625 GETKEYC$
2630 IFC$="J"THENSE=SE+1:GOTO2500
2635 A(S)=SE
2640 NEXTS
2999 :
3000 REM ZAHL ZWISCHEN 0 UND 36 ERZEUGEN
3001 :
3002 FORI=1TOSP:CHAR,0,I*4+1,"            ":CHAR,0,I*4+1,STR$(K(I)):NEXT
3009 SPRITE 1,0
3010 GRAPHIC1
3015 GOSUB6000
3020 COLLISION 2,3300
3050 SPRITE 2,1
3060 MOVSPR2,149,88
3065 R=33:N=R
3100 FORW=(3*π)/2TO(7*π)/2 STEP .3
3105 X=184+R*SIN(W)
3110 Y=88-R*COS(W)
3115 MOVSPR2,X,Y
3120 R=N-.05:N=R
3125 IF R<22ANDRND(1)<.7THEN3500
3130 IFR<19THEN3500
3140 NEXT
3170 GOTO3100
3300 IF BUMP(2)<>2 THEN RETURN
3305 R=R-(RND(1)-.2)
3310 IFN>34 THENN=34
3315 N=R-(RND(1)-.2)
3320 RETURN
3500 SLEEP 2
3505 SPRITE 2,0
3510 SLEEP 1
3515 COLLISION 2
```

```
3900 ZU= RND(-TI)
3915 ZU=INT(RND(1)*37)
3919 FORJ=0TO5
3920 FORI=1TO16
3925 COLOR1,I
3930 CHAR,10,5,STR$(ZU):CHAR,26,5,STR$(ZU)
3935 NEXTI,J
3940 COLOR1,2
3945 CHAR,10,5,STR$(ZU):CHAR,26,5,STR$(ZU)
3950 SLEEP3
3955 CHAR,10,5,"    ":CHAR,26,5,"    "
3960 SPRITE2,0
3999 :
4000 REM AUSWERTUNG
4005 :
4010 FORI=1TOSP
4015 FORJ=1TOA(I)
4020 :SL=INSTR(A$(I,J),"D")
4025 Z$=MID$(A$(I,J),3,SL-3)
4030 FORH=1 TO(LEN(Z$)-1)STEP2
4035 IFZU=VAL(MID$(Z$,H,2))THEN4060
4040 NEXTH
4045 NEXTJ
4050 NEXTI
4055 GOTO5000
4060 G=VAL(MID$(A$(I,J),1,2))
4065 DM=VAL(MID$(A$(I,J),SL+2,(LEN(A$(I,J))-SL)+1))
4070 K(I)=K(I)+S(G)*DM
4075 GOTO4045
4999 :
5000 REM ANZEIGE DES KAPITALS
5005 :
5010 GOSUB6100
5015 FORI=1TOSP
5020 CHAR,0,I*4,NS$(I)+":"
5025 CHAR,0,I*4+1,STR$(K(I))
5030 NEXTI
5035 GOSUB6000
5036 GOSUB10000 :REM EIGENE ERWEITERUNG AUFRUFEN
5040 CHAR,31,5,"STEIGT":CHAR,31,6,"EIN"
5045 CHAR,31,7,"SPIELER":CHAR,31,8,"AUS ?"
5046 PV=0
5047 FORK=1TOSP
5048 IFK(K)=0THENA$="J":PV=1:GOTO5055
5049 NEXTK
5050 GETKEYA$
5055 IFA$="J"THEN5070
5060 IFA$<>"N"THEN5050
5065 GOTO1100
5070 GOSUB6000
5075 CHAR,31,3,"SEIN":CHAR,31,4,"NAME ?"
5076 IFPV=1THENCHAR,31,6,NS$(K):B$=NS$(K):GOTO5115
5080 P2=6:H1$="A":H2$="┘":GOSUB6200
5115 S=1
5120 IFB$=NS$(S)THEN5145
5125 IFS<SPTHENS=S+1:GOTO5120
5130 CHAR,31,15,"DIESEN":CHAR,31,16,"SPIELER":CHAR,31,17,"KENNE ICH"
5135 CHAR,31,18,"NICHT!"
5140 SLEEP5:GOTO5035
5145 FORI=STOSP
5150 NS$(I)=NS$(I+1)
5155 K(I)  =K(I+1)
5160 NEXTI
```

```
5165 SP=SP-1:IFSP=0THENGOTO9000
5170 GOTO5000
5999 :
6000 REM RECHTEN GRAFIKBEREICH LOESCHEN
6005 :
6010 FORI=0TO24
6015 CHAR,31,I,"         "
6020 NEXTI
6025 RETURN
6030 :
6100 REM LINKEN GRAFIKBEREICH LOESCHEN
6105 :
6110 FORI=0TO24
6115 CHAR,0,I,"         "
6120 NEXT
6125 RETURN
6199 :
6200 REM EINGABE EINES STRINGS HOLEN
6201 :
6205 P1=31:B$=""
6210 GETKEYA$
6215 IF((A$)=H1$ANDA$(=H2$)ORA$=" ")ANDP1(=38THEN BEGIN
6220 CHAR,P1,P2,A$:P1=P1+1:B$=B$+A$:BEND
6225 IFA$=CHR$(13) THEN RETURN
6230 IFA$=CHR$(20) AND P1>31 THEN BEGIN
6235 P1=P1-1:CHAR,P1,P2," ":B$=LEFT$(B$,LEN(B$)-1 ):BEND
6240 GOTO 6210
6999 :
7000 REM FEHLERBEHANDLUNG
7001 :
7010 SLOW:GRAPHIC0
7015 COLOR0,12:COLOR5,14
7020 PRINT:PRINTERR$(ER)" ERROR  IN "EL
7025 FORI=1TO8:SPRITEI,0:NEXTI
7030 HELP
7040 END
7499 :
7500 REM ERMITTLLUNG VON SATZ AUS GRAFIK
7501 :
7505 X=RSPPOS(1,0):Y=RSPPOS(1,1)
7510 IFX>154ANDX<204ANDY>130ANDY<224THEN BEGIN
7515 :E$="0"+LEFT$(X$((Y-131)/4,(X-155)/8),1)
7520 :F$=RIGHT$(X$((Y-131)/4,(X-155)/8),2)
7525 :F$=S$(VAL(E$),VAL(F$)):RETURN
7530 BEND
7540 IFX=175ANDY=127THENE$="01":F$="0":RETURN
7545 IFX=134ANDY=139THENE$="08":F$=S$(8,2):RETURN
7550 IFX=120ANDY=185THENE$="09":F$=S$(9,2):RETURN
7555 IFX=115ANDY=215THENE$="10":F$=S$(10,1):RETURN
7560 IFX=115ANDY=229THENE$="06":F$=S$(6,1):RETURN
7565 IFX=130ANDY=230THENE$="06":F$=S$(6,2):RETURN
7570 IFX=145ANDY=231THENE$="06":F$=S$(6,3):RETURN
7575 IFX=162ANDY=232THENE$="07":F$=S$(7,1):RETURN
7580 IFX=178ANDY=232THENE$="07":F$=S$(7,2):RETURN
7585 IFX=194ANDY=232THENE$="07":F$=S$(7,3):RETURN
7590 IFX=214ANDY=218THENE$="10":F$=S$(10,2):RETURN
7595 IFX=232ANDY=187THENE$="09":F$=S$(9,1):RETURN
7600 IFX=218ANDY=137THENE$="08":F$=S$(8,1):RETURN
7610 RETURN
8000 REM ANZEIGE AUS GRAFIK
8001 :
8005 ZE=5
8010 IFVAL(E$)>7THEN8030
```

```
8015 IFLEN(N$(VAL(E$)))>8THENBEGIN: CHAR,31,2,"TRANSVER."
8020 CHAR,32,3,RIGHT$(N$(VAL(E$)),6):GOTO8030
8025 BEND
8026 CHAR,32,2,N$(VAL(E$))
8030 FORI=1TOLEN(F$) STEP 6
8035 CHAR,32,ZE,MID$(F$,I,2)+" "+MID$(F$,I+2,2)+" "+MID$(F$,I+4,2)
8037 ZE=ZE+1
8040 NEXTI
8045 RETURN
8999 :
9000 REM SPIELENDE
9001 :
9010 GOSUB6000:GOSUB6100
9020 CHAR,0,2,"WIR":CHAR,0,3,"DANKEN":CHAR,0,4,"IHNEN"
9025 CHAR,0,5,"FUER":CHAR,0,6,"IHREN":CHAR,0,7,"BESUCH."
9030 CHAR,0,9,"AUF":CHAR,0,10,"WIEDERSEHEN"
9040 SLEEP10:GRAPHIC0
9050 END
9999 :
10000 REM EIGENE ERWEITERUNG BEGINNT HIER UND ENDET MIT RETURN-BEFEHL
10001 :
10010 RETURN
```

Das Programm kann neben der Tastatur auch teilweise über einen Joystick in Port 2 bedient werden.
Dieses Programm schaltet nach dem Start (RUN»ROULETTE« lädt es von Diskette und startet auch automatisch) auf den FAST-Modus, weshalb der Bildschirminhalt für etwa 15 Sekunden nicht sichtbar ist. In dieser Zeit werden die benötigten Variablen eingelesen und die aufwendige Bildschirmgrafik aufgebaut.
Sollte das Programm im 80-Zeichen-Modus, in dem es nicht funktionsfähig ist, gestartet worden sein, wird Ihnen mitgeteilt, daß das Programm auf dem 40-Zeichen-Bildschirm arbeitet; schalten Sie also diesen an, wenn eine entsprechende Meldung erscheint.
Es können maximal 5 Spieler teilnehmen. Steigt ein Spieler ein, muß er Name und Startkapital eingeben. Nach jedem Spiel können ein oder mehrere Spieler ein- oder aussteigen. Spielt kein Spieler mehr mit, wird das Programm beendet.
Hat ein Spieler sein ganzes Geld verloren, so scheidet er automatisch aus. Wenn alle Spieler mit den Eingaben fertig sind, wird das Spiel gestartet. Es erscheinen der Name des Spielers, der dann auswählen muß, worauf er wieviel setzt, und die Frage »Menü oder Tableau?« (Menü = Steuerung über Tastatur, Tableau = Spieltisch, gesteuert über Joystick).
Jedem Spieler stehen also 2 Wahlmöglichkeiten zur Verfügung, <M> oder <T>. Wird die Taste <M> für »Menü« gedrückt, wählt der Spieler damit die Tastatur als Eingabegerät. Dazu werden ihm zuerst alle Sätze angeboten. Nach Wahl eines Satzes kommen auf den Bildschirm die jeweils möglichen Zahlenkombinationen. Mit <NO SCROLL> kann ein eventuelles Scrolling aus dem Bildschirm verhindert werden, da diese Taste die Bildschirmausgabe zunächst stoppt (nochmaliges Drücken von <NO SCROLL> setzt das Programm fort). Der Spieler kann sich dann in Ruhe alle Kombinationen anschauen. Jede Zahlenkombination ist mit einer »Menü-Zahl« versehen, die anschließend bei der Auswahl einzugeben ist und die Zahlenkombination repräsentiert.
Eine weitere Möglichkeit, dies anzuwählen, besteht darin, über Joystick auf dem Tableau eine Auswahl zu treffen (bei der Frage »Menü oder Tableau?« <T> drücken). Man bewegt dazu

einen Chip (Spielmarke) mit dem Joystick über den Bildschirm. An der rechten Seite werden, wenn sich der Joystick im Ruhezustand befindet, der Satz und die Zahlenkombination angezeigt, auf der sich der Chip befindet. Durch Drücken des Feuerknopfes wählt man diese Zahlenkombinationen an.
In beiden Auswahlmöglichkeiten (Menü oder Tableau) kommt anschließend die Frage, wieviel man auf diese Zahl bzw. Zahlenkombination setzen will. Die eingegebene Summe darf das Spielerkapital selbstverständlich nicht übersteigen (andernfalls wird die Eingabe abgelehnt). Nach der Wahl wird der Spieler gefragt, ob er es bei den bis dahin gewählten Zahlenkombinationen beläßt oder auf noch mehr Zahlen setzen will. Haben sich alle Spieler entschieden, »geht nichts mehr« und die Kugel rollt. Die Zahl, die gewinnt, wird angezeigt, und die Spieler, die auf diese Zahl in irgendeiner Weise gesetzt haben, gewinnen den ihrer Chance entsprechenden Betrag ihres Einsatzes.

Grundbegriffe zu Roulette

Zum Verständnis des Programms ist es wichtig, wenigstens die Grundbegriffe von Roulette zu kennen.
Als »Satz« bezeichnet man jede von 13 Arten, auf Zahlen zu setzen:

Plein	1 Zahl
Cheval	2 Zahlen
Transversale plein	3 Zahlen
Carre	4 Zahlen
Transversale simple	6 Zahlen
Dutzend	1-12, 13-24 oder 25-36
Kolonnen	die 3 auf dem Tableau untereinander angeordneten Zahlenreihen, zum Beispiel: 1,4,7,10,13,16,19,22,25,28,31,34
Manque	1-18
Passe	19-36
Impair	ungerade Zahlen
Pair	gerade Zahlen
Rote Zahlen	
Schwarze Zahlen	

Die einzelnen Möglichkeiten eines Satzes (bei »Transversale plein« gibt es beispielsweise 14 davon) werden als »Zahlenkombination« bezeichnet.
Im Programm sind die 13 möglichen Sätze in 10 zusammengefaßt, da »Schwarz« und »Rot«, »Manque« und »Passe« sowie »Impair« und »Pair« als Sätze mit zwei möglichen Zahlenkombinationen behandelt werden.

Arbeitsweise des Programms

Jeder Satz hat eine Nummer, unter der er behandelt wird (Tabelle 3.11):

Tabelle 3.11: *Interne Numerierung der Sätze bei »Roulette«*

1 = Plein
2 = Cheval
3 = Transversale plein
4 = Carre
5 = Transversale simple
6 = Dutzend
7 = Kolonnen
8 = Manque/Passe
9 = Impair/Pair
10 = Rot/Schwarz

Folgende Variablenfelder (Arrays) sind nach dieser Numerierung angeordnet:

- N$(x) enthält den Namen des Satzes »x«
- S (x) enthält den Multiplikator des Satzes »x« für den Gewinnfall
- S$(x,y) enthält die Zahlenkombination »y« zum Satz »x«

Der Multiplikator S(x) ist also der Wert, mit dem der Einsatz multipliziert werden muß, um den Gewinn zu erhalten. Hier ist bereits - im Gegensatz zum richtigen Roulette, bei dem der Einsatz noch hinzugezählt wird - dieser bereits enthalten.
Anmerkung: Jeder Satz hat eine gewisse Chance, die mit der Anzahl der Zahlen steigt, auf die mit einer Zahlenkombination gesetzt werden kann. Die Chance bei »Plein« ist zum Beispiel sehr gering (1:36), bei »Rot« oder »Schwarz« hingegen sehr hoch (1:2). Der Chance entsprechend errechnet sich der Multiplikator: wie bei Glücksspielen üblich, wird bei niedriger Chance eine höhere Gewinnsumme ausbezahlt als bei hoher Gewinnchance. So gewinnt man bei »Plein« das 18fache von einem eventuellen »Rot«-Gewinn, da die Chance bei »Plein« 18mal kleiner ist. Risikobereitschaft soll also honoriert werden.
Die Zahlenkombination »y« des Satzes »x« ist zum Beispiel S$(5,11)=»313233343536«, das heißt S$(5,11) enthält die Zahlenkombination Nummer 11 des Satzes »Transversale simple«. Jede Zahl dieser Kombination hat genau 2 Ziffern (bei Zahlen unter 10 steht eine führende Null).

Aufbau der DATA-Zeilen

In den DATA-Zeilen muß man nun je einen Satz als Block betrachten, z.B. die Zeilen 105-109. Die erste Zahl in 105 ist die Anzahl der Zahlenkombinationen des Satzes - in diesem Fall »Cheval« -, also ist Z(2) diese erste Zahl (2 ist die Nummer für »Cheval«).
Die dann folgenden Daten werden in S$(x,y) eingelesen. Auf sie folgt der Multiplikator des Einsatzes, im Beispiel S(2), und als Abschluß der Name des Satzes, im Beispiel n$(2). Von Zeile 200 bis 240 werden diese Arrays eingelesen.
Aufschlüsselung des Programms nach Zeilennummern (Tabelle 3.12):

Tabelle 3.12: *Aufschlüsselung von »Roulette« nach Zeilennummern*

Zeilen	Inhalt
0 - 30:	Initialisierung
100 - 146:	Daten für Sätze
150 - 172:	Daten für Joystick-Steuerung
200 - 299:	Einlesen der Daten
300 - 999:	Zeichnen der Grafik
1000 - 1999:	Eingabe von Name und Kapital des Spielers
2000 - 2999:	Eingabe der Spieler, worauf sie setzen.
2030 - 2499:	Eingabe mit Joystick
2500 - 2999:	Eingabe via Menü
3000 - 3999:	Grafische Darstellung der rollenden Kugel, Erzeugung einer Zufallszahl zwischen 0 und 36
4000 - 4999:	Auswertung und Aktualisierung des Spielerkapitals
5000 - 5999:	Anzeige des Kapitals, Spielausstieg, neue Runde
6000 - 6099:	Löschen des rechten Grafikbereichs
6100 - 6199:	Löschen des linken Grafikbereichs
6200 - 6999:	Eingabe von Name und Kapital im rechten Grafikteil H1$ = kleinstes zugelassenes Eingabezeichen H2$ = größtes zugelassenes Eingabezeichen P2 = Zeile, bei der die Eingabe beginnt B$ = Eingabestring (nach Ende der Routine)
7000 - 7499:	erweiterte Fehlerbehandlungsroutine Löschen aller Sprites und Umschalten auf SLOW vor Ausgabe der Fehlermeldung über PRINT
7500 - 7599:	Ermittlung des Satzes und der Zahlenkombination aus der Position des Chips, der mit dem Joystick gesteuert wird (Zeilen 2030-2499) E$ = Satz F$ = Zahlenkombination
8000 - 8999:	Anzeige von Satz und Zahlenkombination im rechten Grafikbereich
9000 - 9999:	Spielende
10000 - xxxx:	eigene Erweiterung, die mit RETURN-Befehl endet

Im Programm »Roulette« befinden sich also einige interessante Routinen, die eventuell für eine Weiterverwendung in eigenen Programmen geeignet sind, wie etwa die Eingaberoutine oder das schnelle Löschen eines Grafikbereichs.
Auch Zeile 10 ist interessant, wenn ein Programm nur im 40-Zeichen-Modus läuft und eine Warnung erfolgen soll, falls sich der Computer zum Zeitpunkt des Programmstarts im 80er-Modus befindet.

3.6.3 Gerät verfügbar? – Prüfroutine aufrufen!

Da das Fehlen oder Nicht-angeschaltet-Sein von Floppy oder Drucker verhängnisvolle Auswirkungen (fehlende Absicherung von Daten, Abbruch des Programms durch Fehlermeldung usw.) haben kann, sollten einigermaßen bedienungsfreundliche Programme von sich aus feststellen, ob ein Gerät verfügbar ist. Dies kann in Basic 7.0 dank der Befehle TRAP und RESUME einfach programmiert werden, da sich die programmierte Fehlerbehandlung auch eignet, um »?DEVICE NOT PRESENT ERROR« abzufangen.
Zunächst wollen wir für das Diskettenlaufwerk (Gerätenummer 8) ein solches Beispiel durchgehen.
Beschreibung: Prüfen, ob Floppy (Gerätenummer 8) verfügbar ist
Filename: »floppy pruefen«

```
10 TRAP 50000
20 OPEN1,8,15,"I":CLOSE1
30 END
40 :
50000 REM FEHLERBEHANDLUNG
50010 PRINT "BITTE DIE FLOPPY EINSCHALTEN ! <TASTE>":GETKEY A$:RESUME
```

Besprechen wir dieses kurze Programm.
In Zeile 10 wird die eigene Fehlerbehandlungsroutine (Zeilen 50000/50010) aktiviert. Dann kann in Zeile 20 die Floppy angesprochen werden; ist die Floppy angeschaltet, wird der Initialisierungsbefehl ausgeführt, ansonsten stellt Basic 7.0 dies automatisch fest und will »?DEVICE NOT PRESENT ERROR« ausgeben; da durch TRAP 50000 die Basic-7.0-Fehlerbehandlung ausgeschaltet wurde, wird zu Zeile 50000 verzweigt.
In Zeile 50010 ergeht dann die Aufforderung an den Benutzer, das Diskettenlaufwerk anzuschalten. Mit RESUME wird das Programm wieder beim letzten Befehl fortgesetzt.
Anstelle der Initialisierung kann man auch DCLEAR#8 verwenden, was außer einem Test, ob die Floppy verfügbar ist, auch zusätzlich alle auf der Floppy geöffneten Kanäle schließt. Deshalb ist DCLEAR#8 sogar die bessere der beiden Möglichkeiten.
Ähnlich ist es mit folgendem Programm für den Drucker, dem CHR$(0) gesendet wird, damit er kein Zeichen ausgibt, da dies ein unerwünschter Nebeneffekt wäre:
Beschreibung: Prüfen, ob Drucker (Gerätenummer 4) verfügbar ist
Filename: »drucker pruefen«

```
10 TRAP 50000
20 OPEN1,4,0,CHR$(0):CLOSE1
30 END
40 :
50000 REM FEHLERBEHANDLUNG
50010 PRINT "BITTE DEN DRUCKER EINSCHALTEN ! <TASTE>":GETKEY A$:RESUME
```

Hier würde es zwar naheliegen, DCLEAR #4 anstelle von

```
"OPEN1,4,0,CHR$(0):CLOSE1"
```

zu verwenden, was aber nicht möglich ist, da DCLEAR nur für den Betrieb mit Diskettenlaufwerken vorgesehen ist. DCLEAR #4 wird zwar nicht mit einer Fehlermeldung abgewiesen, aber die Wirkung entspricht nicht der gewünschten.

DCLEAR mit Floppy und Drucker

Die Wirkung von DCLEAR, alle Kanäle auf einem Gerät zu schließen, erspart auch später auftretende Fehler der Sorte »?FILE OPEN ERROR« (ein Ersatz für DCLOSE ist dies jedoch nicht, da die Kanäle nur computerintern gelöscht werden, aber Files auf Diskette nicht geschlossen werden; dies hat dann File-Einträge wie »*PRG«, »*SEQ«, oder »*REL« zur Folge).
Zudem wird die Floppy über ihr Kommando »INITIALIZE« (»I«) initialisiert.
Für den Drucker gibt es eine Hilfslösung, um alle auf ihm geöffneten Kanäle (computerintern) zu schließen, die aber ebenfalls kein gleichwertiger Ersatz für CLOSE ist:

```
BANK 15:SYS DEC("FF4A"),4
```

Dieser Befehl muß, soll er in unseren Prüfroutinen auftauchen, vor »OPEN4,4,0,CHR$(0): CLOSE 4« stehen.

Weitere Ergänzungen zu den Prüfroutinen

Der große Mangel der beiden vorgestellten Routinen ist, daß danach bei jedem auftretenden Fehler (auch »?SYNTAX ERROR«, »?ILLEGAL QUANTITY« usw.) die Aufforderung zum Einschalten von Floppy oder Drucker ausgegeben wird, obwohl dies nicht unbedingt verursachend gewesen sein muß. Deshalb kann zusätzlich die Variable ER mit 5 (Code für »?DEVICE NOT PRESENT ERROR«) verglichen werden; ist ER=5, so war ein Gerät nicht verfügbar, ansonsten handelt es sich um einen anderen Fehler.
Am Beispiel der Floppy-Prüfroutine:

```
50010 IF ER=5 THEN PRINT "BITTE DIE FLOPPY EINSCHALTEN!
<TASTE>":GETKEY A$:RESUME:ELSE PRINT "?";ERR$(ER);" IN";EL:END
```

Für die Drucker-Prüfroutine ist nur der auszugebende Text anders.
Die Geräteadressen können übrigens durch Ändern des OPEN- bzw. DCLEAR-Parameters leicht geändert werden.
Eine andere Lösung wäre, ein Flag zu setzen. G soll den Wert 0 haben, wenn das Gerät verfügbar ist, den Wert -1, wenn dies nicht der Fall ist. Dazu sind folgende Änderungen erforderlich (Beispiel der Floppy-Prüfroutine):

```
20 G=0:DCLEAR!8
50010 G=(ER=5):IF G THEN RESUME NEXT:ELSE PRINT "?";ERR$(ER);" IN"; EL:
END
```

Eine Meldung wird dann nicht mehr in der Fehlerbehandlungsroutine selbst ausgegeben, sondern nach Abfrage von G im Hauptprogramm:

```
30 IF G THEN PRINT "FLOPPY ANSCHALTEN! <TASTE>":GETKEY A$:GOTO 20
```

Auch eine DO-LOOP-Warteschleife ist möglich (noch einmal die komplette Routine):

```
10 TRAP 50000
20 DO
```

```
30 :G=0
40 :DCLEAR#8
50 :IF G=0 THEN EXIT
60 :PRINT "FLOPPY ANSCHALTEN! <TASTE>":GETKEY A$
70 LOOP
80 END
90 :
50000 REM Fehlerbehandlungsroutine
50010 G=(ER=5):IF G THEN RESUME NEXT:
ELSE PRINT "?";ERR$(ER);" IN";EL:END
```

Diese Version ist auf der Programmdiskette unter dem Namen »FLOPPY PR.KOMPL.« abgespeichert.
Eine entsprechende Prüfroutine für den Drucker ändert sich nur an folgenden Stellen:

```
40 :BANK15:SYS DEC("FF4A"),4:OPEN 4,4,0,CHR$(0):CLOSE 4
60 :PRINT "DRUCKER ANSCHALTEN! <TASTE>":GETKEY A$
```

Zeile 40 ist die Ersatzkonstruktion für »DCLEAR#4«, Zeile 60 ist die geänderte Meldung (»DRUCKER« statt »FLOPPY«).

3.6.4 Professionelle Eingaberoutine

Der INPUT-Befehl, zu dem auch Basic 7.0 keine Alternative kennt, hat folgende mehr oder weniger gravierenden Mängel:

- Eingabe von Semikolon, Komma und Doppelpunkt nur in Anführungszeichen möglich
- Eingabe von Anführungszeichen wird ignoriert
- keine Beschränkung der Eingabe auf bestimmte Zeichen möglich (»?REDO FROM START«-Meldung ist eine Folge dieses Mangels)
- keine Angabe von Minimal- und Maximallänge der Eingabe
- Steuertasten wie <CLR/HOME> oder Cursortasten werden ausgeführt
- keine Erweiterbarkeit zu einer Eingabemaske
- kein vordefiniertes Eingabefeld

Eingabefeld und Eingabemaske

Diese beiden Begriffe müssen erklärt werden.
Als »Eingabefeld« bezeichnet man einen Bildschirmbereich, in dem eine Eingabe erfolgt. Der INPUT-Befehl kennt nur den ganzen Bildschirm bzw. das definierte Window als Eingabemaske, da man mit den Cursortasten und anderen (auch ESC-Sequenzen werden ausgeführt) den Bildschirm während einer INPUT-Eingabe beliebig editieren kann. Zum Beispiel kann man mit den Cursortasten, ESC-B und ESC-T ein Window definieren, was zu Fehlfunktionen eines Programms, das auf eine Verkleinerung des Bildschirmformats nicht eingerichtet ist,

führt, oder vom 40- in den 80-Zeichen-Bildschirm (oder umgekehrt) mit ESC-X wechseln oder ein extra für die Eingabe vorgesehenes Window mit <HOME><HOME> auflösen usw. usf. Wenn man es also darauf absieht, kann man eine INPUT-Eingabe zu einigen Manipulationen mißbrauchen, um beispielsweise im Programm eine Fehlermeldung zu erzeugen oder nur den Programmablauf zu stören, sei es aus Versehen oder gezielt.
Als »Eingabemaske« bezeichnet man eine Gruppe von gleichzeitig am Bildschirm editierbaren Eingabefeldern, zwischen denen beliebig gewechselt werden kann. Während ein Eingabefeld z.B. den Namen einer Person aufnimmt, kann die gesamte Eingabemaske die komplette Adresse beinhalten, da außer dem Eingabefeld des Namens auch noch Felder für Straße/Hausnummer, Wohnort und Telefonnummer vorhanden sind. Wer einmal mit einer professionellen Dateiverwaltung gearbeitet hat, weiß dies sicher zu schätzen.
Das Problem bei INPUT ist nun, daß man alle Eingaben nacheinander tätigen muß, und zwar genau in der Reihenfolge, wie es das Programm vorschreibt.

So behebt man die Mängel von INPUT

Damit, daß wir das INPUT-Kommando kritisieren, ist es natürlich nicht getan. Wir wollen unter Zuhilfenahme der Basic-7.0-Befehle eine eigene Eingaberoutine programmieren, die die genannten Mängel nicht mehr aufweist. Dabei wird zunächst das Programm besprochen und im Text weiterentwickelt; wenn Ihnen dies an manchen Stellen kompliziert erscheint, so können Sie auch einfach weiterlesen und sich nur die Stellen herauspicken, an denen die Verwendung der fertigen Routine besprochen wird. Die Anwendung der noch vorzustellenden Eingaberoutine ist eine wahre Freude, denn man erleichtert sich die Programmierung auf diese Weise erheblich und kann den Eingabekomfort und die Eingabesicherheit eines Programms steigern, ohne daß man erheblichen Mehraufwand in Kauf nehmen muß. Die Routine ist von Natur aus als Unterprogramm geschrieben und kann problemlos in eigene Programme eingebaut werden.
Unsere Routine soll sich durch folgende Vorzüge auszeichnen:

- Eingabe von Komma, Semikolon, Doppelpunkt und Anführungszeichen
- Forderung von Mindest- und Höchstlänge der Eingabe (z.B. 10-40 Zeichen) wird von der Eingaberoutine unterstützt
- Steuertasten wirken nicht (oder nur im Sinne der Routine)
- fest vordefiniertes Eingabefeld (z.B. Spalten 10-30 in Zeile 5)
- Erweiterbarkeit zur Eingabemaske
- hohe Flexibilität (z.B. können zu <RETURN> weitere Tasten definiert werden, die das Ende der Eingabe bewirken)
- wie bei INPUT kann eine Vorbelegung der Eingabe erfolgen, die dann vom Anwender nur noch durch <RETURN> bestätigt werden muß

Die eigene Eingaberoutine

Jetzt soll Ihnen die lange versprochene Unterroutine endlich vorgestellt werden. Sie stützt sich auf einige Parameter, die vom Hauptprogramm übergeben werden. Zur Demonstration geben Sie bitte folgenden Befehl ein: RUN»DEMO.EINGABE«

Anhand dieses Demonstrationsprogrammes wollen wir die Einstellmöglichkeiten und die Bedienung der Eingaberoutine durch den Anwender besprechen.
Laden und starten Sie also das Demoprogramm wie beschrieben, und entnehmen Sie hier die Beschreibung der einzelnen Parameter.

- *Spalte der Eingabe (Variable S)*

Damit wird die Spalte, ab der die Eingabe beginnen soll, eingestellt. Diese wird von einem CHAR-Befehl weiterverwendet und muß also im Bereich 0-39 (40-Zeichen-Modus) oder 0-79 (80-Zeichen-Modus) liegen.
Im Demoprogramm »DEMO.EINGABE« ist 5 voreingestellt, für diesen Wert muß also nur <RETURN> betätigt werden.

- *Zeile der Eingabe (Variable Z)*

Auch die Zeile, in der die Eingabe erfolgt, wird eingestellt. Die Eingabe kann nicht mehr als eine Zeile, also je nach Darstellungsmodus 40 oder 80 Zeichen, umfassen.
In »DEMO.EINGABE« ist 3 voreingestellt.

- *Mindestlänge der Eingabe (Variable ML)*

Unterschreitet eine Eingabe die vorgeschriebene Mindestlänge, für die in »DEMO.EINGABE« 5 voreingestellt ist, so wird die Eingabe nach Betätigen von <RETURN> nicht angenommen, sondern fortgesetzt, als wenn keine Taste gedrückt worden wäre.

- *Maximallänge der Eingabe (Variable MX)*

Die Maximallänge, für die das Demoprogramm 25 vorsieht, hat Auswirkungen auf die Größe des Eingabefeldes. Dadurch kann die Maximallänge nicht überschritten werden (an der letzten Position kann der Cursor nicht nach rechts bewegt werden).

- *Vorbelegungsflag (Variable VB)*

Die Vorbelegung des Eingabefeldes muß in IN$ stehen. Ist VB=-1, so wird eine Vorbelegung gewünscht, die das Demoprogramm einholt.
Ansonsten muß VB den voreingestellten Wert 0 haben.

- *Erlaubte Zeichen (Variable E$)*

Die bei der Eingabe zulässigen Zeichen (z.B. alle Ziffern) müssen in E$ stehen. Um die Anwendung des Unterprogramms möglichst unkompliziert zu gestalten, ist die Angabe der Steuertasten (<RETURN>, <INST/DEL> usw.) nicht erforderlich.
Der im Demoprogramm voreingestellte Eingabestring enthält alle Buchstaben und Ziffern. Möglich sind auch diejenigen Zeichen, die INPUT nicht zuläßt (Semikolon, Komma, Doppelpunkt, Anführungszeichen). Da das Demoprogramm aber zunächst über INPUT arbeitet, werden Sie Schwierigkeiten bei der Eingabe dieser Zeichen haben. In einem Programm könnte man aber E$ entsprechend belegen (CHR$(34)=Anführungszeichen).

Das Zeichen, das durch Betätigen von <CBM>+<§> erreicht wird, darf als einziges Zeichen nicht in E$ stehen, da dies sonst zu Fehlfunktionen führen kann. Dieses Zeichen dient nämlich als Leerstelle im Eingabefeld, um das Eingabefeld sichtbar zu machen.

– *Zeichen für Eingabe-Ende (Variable ZR$)*

Außer der <RETURN>-Taste, die von der Eingabe zurück ins Hauptprogramm führt, können (müssen aber nicht) noch weitere Tasten definiert werden. ZR$ muß diese Zeichen enthalten; ein <RETURN> wird durch ZR$=CHR$(13) eingestellt (Voreinstellung im Demoprogramm).
Die Möglichkeit mehrerer Tasten dieser Art ist bei der Erweiterung zur Eingabemaske wichtig, was wir zunächst nicht besprechen müssen.
Nach der Eingabe dieser Parameter fordert Sie dann das Demoprogramm zu einer – Ihren Parametereingaben entsprechenden – Eingabe auf. Das soll uns Anlaß sein, auf die Bedienung der Eingaberoutine einzugehen, bevor wir die Funktionsweise zerpflücken.
Wenn ein zulässiges Eingabezeichen eingegeben wird, so wird dieses in den Eingabestring aufgenommen und angezeigt. Mit <INST/DEL> kann man einzelne Zeichen löschen, mit <SHIFT>+<INST/DEL> Platz für Einfügungen schaffen. Durch <RETURN> wird die Eingabe beendet und ins Hauptprogramm zurückgesprungen, sofern die Mindestlänge der Eingabe erreicht ist. Die Leerstellenstriche gelten nicht als eingegebene Zeichen.
Mit <CLR/HOME> kommt man an die erste Position des Eingabefeldes, mit <SHIFT>+<CLR/HOME> löscht man das Eingabefeld. <CRSR LEFT> und <CRSR RIGHT> funktionieren ähnlich wie bei INPUT, der Cursor ist jedoch ein Festcursor (blinkt also nicht) und kann nur innerhalb das durch die Leerstellen markierten Eingabefeldes bzw. über eingegebenen Zeichen bewegt werden. Wenn der Cursor also an einer bestimmten Position stehenbleibt, so ist dies kein Programmfehler, sondern völlig korrekt.
Mit <CRSR DOWN> löscht man alle Zeichen ab der Cursorposition (einschließlich des Zeichens an der Cursorposition selbst), mit <CRSR UP> löscht man alle Zeichen bis zur Cursor-
Bei <CRSR UP> wird der verbleibende Teil der Eingabe automatisch an den Anfang des Eingabefeldes gezogen.
Andere Tasten als die genannten werden ignoriert.
Am besten probieren Sie die Eingaberoutine erst einmal aus, und wenn Sie dann die Bedienung verstanden haben, lesen Sie hier in der Programmbeschreibung weiter. Wollen Sie die Routine nicht verstehen, sondern nur in eigenen Programmen weiterverwenden, finden Sie am Ende dieses Unterkapitels 3.6.4 alle nötigen Informationen; es sei Ihnen dennoch dazu angeraten, die Programmbeschreibung zu lesen, denn die Routine funktioniert auf einfachere Weise, als es ihre Leistung vermuten läßt.
Achtung: Diese Routine darf nicht einfach über RUN gestartet werden; das Programm ist nur lauffähig, wenn vorher alle erforderlichen Parameter mitgeteilt wurden, wie dies im Demoprogramm der Fall ist.

Beschreibung: Professionelle Eingaberoutine als Unterprogramm ab Zeile 50000
Filename: »eingabe.50000«

```
50000 REM *****************************
50010 REM ***                       ***
50020 REM ***    EINGABEROUTINE     ***
50030 REM ***    ==============     ***
50040 REM ***                       ***
50050 REM *****************************
50060 REM
50070 REM UEBERGABEPARAMETER
50080 REM ==================
50090 REM
50100 REM VB = VORBELEGUNGSFLAG FUER EINGABE (-1 = EINGABESTRING VORBELEGT)
50110 REM IN$= EINGABESTRING (NUR BEI VB=-1)
50120 REM E$ = STRING DER ERLAUBTEN ZEICHEN (WIRD ERWEITERT ZU E1$)
50130 REM MX = MAXIMALE LAENGE DER EINGABE  (= ANZAHL DER FUELLZEICHEN)
50140 REM ML = MINIMALE LAENGE DER EINGABE  (WIRD BEI EINGABE-ENDE UEBERPRUEFT)
50150 REM Z  = ZEILE FUER EINGABE           (FUER CHAR-BEFEHL)
50160 REM S  = SPALTE (ANFANGSSPALTE) FUER EINGABE (FUER CHAR-BEFEHL)
50170 REM ZR$= STRING DER TASTEN FUER BEENDEN DER EINGABE
50180 REM
50190 REM BERECHNUNG WEITERER WERTE AUS DEN PARAMETERN
50200 REM ============================================
50210 :
50220 P=1:REM POSITION INNERHALB DES STRINGS
50230 E1$=E$+ZR$+CHR$(17)+CHR$(19)+CHR$(20)+CHR$(29)+CHR$(145)+CHR$(147)+CHR$(14
8)+CHR$(157):REM ERLAUBTE ZEICHEN ERWEITERN
50240 REM BERECHNUNG DES EINGABESTRINGS
50250 IF NOT VB THEN IN$="":FOR F=1 TO MX:IN$=IN$+"_":NEXT
50260 IF VB THEN LZ=MX:DO UNTIL LEN(IN$)=MX:IN$=IN$+"_":LOOP:GOTO 50330
50270 :
50280 REM AKTUELLE WERTE BERECHNEN
50290 REM ========================
50300 :
50310 LZ=2
50320 :
50330 FL=(P=MX):REM FLAG FUER LETZTE POSITION
50340 FE=(P= 1):REM FLAG FUER ERSTE POSITION
50350 :
50360 DO WHILE MID$(IN$,LZ,1)="_" AND LZ>1:LZ=LZ-1:LOOP:REM LZ BERECHNEN
50370 :
50380 REM AUSGABE DER STRINGS
50390 REM ===================
50400 :
50410 CHAR,S,Z,IN$
50420 CHAR,S+P-1,Z,MID$(IN$,P,1),1:REM FEST-CURSOR ERZEUGEN
50430 :
50440 REM EINGABESCHLEIFE
50450 REM ===============
50460 :
50470 DO:GETKEY A$:LOOP UNTIL INSTR(E1$,A$)
50480 :
50490 IF INSTR(E$,A$) THEN BEGIN:REM EINGABEZEICHEN IN STRING AUFNEHMEN
50500 :MID$(IN$,P,1)=A$
50510 :P=P+1+(P=MX):LZ=LZ+1
50520 :GOTO 50330
50530 BEND
50540 :
50550 :
50560 IF INSTR(ZR$,A$) THEN BEGIN:REM EINGABE-ENDE
50570 :IF LZ<ML THEN GOTO 50470
```

```
50580 :IN$=LEFT$(IN$,LZ):H$=IN$:IN$=""
50590 :FOR F=1 TO LZ
50600 :IF MID$(H$,F,1)<>"_"THEN IN$=IN$+MID$(H$,F,1)
50610 :NEXT:IF LEN(IN$)<ML THEN IN$=H$:GOTO 50470
50620 :RETURN
50630 BEND
50640 :
50650 IF A$=CHR$(157) THEN IF NOT FE THEN P=P-1:GOTO 50330:REM CURSOR LINKS
50660 :
50670 IF A$=CHR$(147) THEN IN$="":GOTO 50220:REM CLR HOME
50680 :
50690 IF A$=CHR$(29) THEN P=P-((P<>MX) AND (P<>LZ+1)):GOTO 50330:REM CRSR RECHTS
50700 :
50710 IF A$=CHR$(145) THEN BEGIN:REM CRSR NACH OBEN
50720 :IF P=1 THEN 50470
50730 :H$=IN$:IN$=""
50740 :FOR F=P TO MX:IN$=IN$+MID$(H$,F,1):NEXT:LZ=LEN(IN$):DO UNTIL LEN(IN$)=MX:
IN$=IN$+"_":LOOP:P=1
50750 :GOTO 50330
50760 BEND
50770 :
50780 IF A$=CHR$(17) THEN BEGIN:REM CRSR NACH UNTEN
50790 :FOR F= P TO MX:MID$(IN$,F,1)="_":NEXT:LZ=P
50800 :GOTO 50330
50810 BEND
50820 :
50830 IF A$=CHR$(20) THEN BEGIN:REM DELETE (DEL)
50840 :IF FE THEN 50470
50850 :IN$=LEFT$(IN$,P-2)+RIGHT$(IN$,MX-P+1)
50860 :DO UNTIL LEN(IN$)=MX:IN$=IN$+"_":LOOP
50870 :P=P-1
50880 :GOTO 50330
50890 BEND
50900 :
50910 IF A$=CHR$(148) THEN BEGIN:REM INSERT (INST)
50920 :IF LZ=MX THEN 50470
50930 :IF FL THEN 50470
50940 :IN$=LEFT$(IN$,P-1)+"_"+"RIGHT$(IN$,MX-P+1):IN$=LEFT$(IN$,MX)
50950 :LZ=LZ+1
50960 :GOTO 50330
50970 BEND
50980 :
50990 IF A$=CHR$(19) THEN P=1:GOTO 50330:REM HOME
51000 :
51010 GOTO 50470:REM ZURUECK ZUR EINGABESCHLEIFE
```

Die Routine liegt im Zeilenbereich ab 50000, damit sie als Unterroutine am Programmende stehen kann.

Erklärung der Eingaberoutine

Zunächst müssen wir diejenigen Variablen des Programms besprechen, die nicht als Parameter übergeben werden.

- *Variable IN$: Eingabestring*

Dieser Eingabestring enthält vor Aufruf der Routine die Vorbelegung (wenn VB=-1 ist), während der Eingabe den am Bildschirm dargestellten Eingabestring (mit allen Leerstellen usw.)

und beim Rücksprung ins Hauptprogramm die ausgewertete Eingabe (ohne Leerstellen, nur gültige Zeichen).

- *Variable P: Position innerhalb des Eingabestrings*

Innerhalb des Eingabestrings wird der Festcursor (nicht blinkender Cursor) dargestellt. Dazu muß die Cursorposition innerhalb von IN$ festgehalten werden; sie steht in der Variablen P.

- *Variable E1$: Erweiterung von E$*

Der String E$, der die vom Hauptprogramm zugelassenen Zeichen enthält, wird am Anfang der Routine zu E1$ erweitert und enthält dann außer E$ noch alle Zeichen von ZR$ (Zeichen für Eingabe-Ende) und die Steuertasten der Eingaberoutine wie <INST/DEL> oder die Cursortasten.

- *Variable LZ: Position des letzten Zeichens in IN$*

Da IN$ während der Eingabe auch alle Leerstellen enthält, muß jederzeit festgestellt werden können, an welcher Position das letzte gültige Zeichen (= Nicht-Leerstelle) steht. Diese Variable wird nach jeder Eingabe eines Zeichens aktualisiert, da sie für das fehlerfreie Funktionieren sehr wichtig ist.

- *Variable FL: Flag für letzte Position*

Wenn der Cursor an der letzten Position des Eingabefeldes steht, ist FL=-1, ansonsten 0.

- *Variable FE: Flag für erste Position*

FE=-1: Cursor an erster Position im Eingabefeld
FE= 0: Cursor nicht an erster Position

- *Variable A$: Taste, die zuletzt gedrückt wurde*

Die letzte betätigte Taste, die über GETKEY abgefragt wurde, wird in A$ aufbewahrt und zu Vergleichszwecken während der Bearbeitung dieser Taste wiederholt durch IF...THEN abgefragt.

- *Variable F: Schleifenzähler*

Die Variable F wird als Schleifenzähler verwendet. Da die meisten Programmierer als Schleifenzähler die Variable I einsetzen, habe ich bei der Programmierung der Eingaberoutine »F« verwendet, damit es keine Überschneidungen von Schleifenvariablen gibt, wenn die Eingaberoutine in anderen Programmen verwendet wird.
Ihre Programme, die diese Eingaberoutine enthalten, können also ohne Bedenken die Variable I als Schleifenzähler verwenden.

- *Variable H$: Hilfsstring*

Bei Beenden der Eingabe werden aus IN$ alle Leerstellen (<CBM>+<§>) entfernt. Dafür wird H$ als Zwischenspeicher benutzt.

Aufschlüsselung des Programms nach Zeilennummern

Da in »EINGABE.50000« (Listing 23) sehr viele REM-Kommentare stehen und das Programm dank vieler Einrückungen äußerst übersichtlich ist, fehlt zur Analyse dieser Routine noch der nach Zeilennummern geordnete Überblick als Tabelle 3.13. Besonders interessante oder komplizierte Stellen des Programms werden wir noch gesondert besprechen.

Tabelle 3.13: *Aufschlüsselung der Eingaberoutine nach Zeilennummern*

50000-50180:	REM-Zeilen mit Kommentaren zu den Parametern
50190-50270:	Berechnung/Initialisierung der Variablen P,E1$,IN$ IN$ wird gemäß dem Vorbelegungsflag VB gesetzt: VB=-1: IN$ wird nur mit Leerstellen aufgefüllt VB= 0: IN$ wird auf die Maximallänge gebracht, indem Leerstellen angehängt werden
50280-50370:	Variablen LZ,FL und FE aktualisieren (die Aktualisierung von LZ wird noch besprochen)
50380-50430:	Eingabestring ausgeben und Festcursor erzeugen, indem das Zeichen an Cursorposition noch einmal ausgegeben wird, aber dann revers (Zeile 50420)
50440-50480:	Eingabeschleife; auf in E1$ enthaltenes Zeichen mit DO-LOOP-Schleife warten (siehe auch 3.4.6)
50490-50550:	In E$ enthaltenes und eingegebenes Zeichen in den Eingabestring IN$ mittels »MID$(...)=« aufnehmen
50560-50640:	Eingabe-Ende, Prüfung auf Mindestlänge
50650-50660:	Cursor-Links-Bewegung ausführen
50670-50680:	Eingabefeld löschen nach <SHIFT>+<CLR/HOME>
50690-50700:	Cursor-Rechts-Bewegung ausführen
50710-50770:	Eingabefeld bis Cursorposition löschen (CRSR UP)
50780-50820:	Eingabefeld ab Cursorposition löschen (CRSR DOWN)
50830-50900:	Zeichen löschen (DEL)
50910-50980:	Zeichen einfügen (INST = <SHIFT>+<INST/DEL>)
50990-51000:	Cursor an Anfang des Eingabefeldes bewegen (HOME)
51010 :	Rücksprung zur Eingabeschleife

Besonders interessante oder komplizierte Programmteile

Da die Eingaberoutine an einigen Stellen sehr trickreich programmiert ist, um möglichst effektiv zu arbeiten, greifen wir jetzt einige Programmzeilen heraus und analysieren diese.

- *Zeilen 50330/50340: Flags berechnen*

Die Flags, ob der Cursor an erster oder letzter Position des Eingabefeldes steht, werden von den Programmteilen zur Cursorbewegung nach rechts oder links abgefragt: Von der ersten Position aus ist keine Links-, von der letzten keine Rechtsbewegung möglich. Dadurch wird ein Verlassen des Eingabefeldes verhindert.
Diese Flags werden nach der in 3.3 vorgestellten Methode zuerst berechnet, das Ergebnis wird in den Variablen FL und FE gespeichert und erst später über IF abgefragt.

- *Zeile 50360: Aktualisierung von LZ*

In LZ steht die Position des letzten gültigen Zeichens innerhalb des Eingabestrings IN$. Dieser Wert ändert sich mit fast jedem Tastendruck, wenn man von den Cursorbewegungen oder einigen Spezialfällen der Zeicheneingabe absieht.
Mit Hilfe einer DO-LOOP-Schleife wird LZ berechnet. Vor dieser DO-LOOP-Schleife in Zeile 50360 muß LZ den maximalen Wert enthalten, da es in der Schleife heruntergezählt, aber auf gar keinen Fall erhöht wird. Deshalb wird LZ von den einzelnen Programmteilen, wie beispielsweise der Texteingabe (50490-50550), gerade soweit erhöht, wie sich der Wert von LZ im Extremfall geändert hat. Beider Eingabe eines einzelnen Zeichens wird beispielsweise 1 addiert (Zeile 50510), da nur jeweils ein einziges eingegebenes Zeichen behandelt wird.
Kommen wir auf die DO-LOOP-Schleife in Zeile 50360 zurück. Diese wird – unter Umständen ohne einen einzigen vorher erfolgten Schleifendurchlauf – abgebrochen, wenn an der Position von LZ ein anderes Zeichen als die Leerstelle <CBM>+<§> steht. Andernfalls wird LZ um 1 verringert, da das letzte gültige Zeichen folglich weiter hinten im String IN$ liegen muß. Bei LZ=1 wird abgebrochen, da 1 der Minimalwert sein soll (»AND LZ>1« im WHILE-Ausdruck sorgt dafür).

- *Zeile 50470: auf zulässigen Tastendruck warten*

In Zeile 50470 wird mit Hilfe einer DO-LOOP-Schleife so lange auf einen Tastendruck gewartet, bis dieser (A$) eines der in E1$ enthaltenen (zulässigen) Zeichen ist. Interessant ist dabei, daß der in 3.4.6 beschriebene Trick eingesetzt wird: Wenn A$ in E1$ enthalten ist, liefert INSTR(E1$,A$) einen Wert ungleich 0, der nach UNTIL als wahr gilt und somit zum Beenden der Schleife führt. Kommt A$ nicht in E1$ vor, so ist INSTR(E1$,A$) dem Wert nach 0 und führt hinter UNTIL zum Fortsetzen der Schleife, da 0 für einen falschen Ausdruck steht.
Im Prinzip wird nur der Vergleich mit 0 gespart. Statt

```
DO:GETKEY A$:LOOP UNTIL INSTR(E1$,A$)
```

könnte es auch

```
DO:GETKEY A$:LOOP UNTIL INSTR(E1$,A$)<>0
```

heißen, was aber eingespart werden kann und die Verarbeitungsgeschwindigkeit erhöht.

- *Zeile 50510: Beispiel für die Aktualisierung von P und LZ*

Wenn der Eingabestring verändert wird (Eingabe eines Zeichens, Löschen eines oder mehrerer Zeichen), so muß die Cursorposition P entsprechend aktualisiert werden. In Zeile 50510, die zur Routine »Eingabe eines Zeichens aus E$« gehört, wird dabei der in 3.3 beschriebene Programmiertrick eingesetzt:

```
P=P+1+(P=MX)
```

Der Cursor wird zunächst um 1 erhöht, sollte die Position aber die letzte im Eingabefeld gewesen sein (»(P=MX)«), so wird wieder 1 abgezogen.
Mit IF...THEN würde man dies wie folgt schreiben:

```
IF P<>MX THEN P=P+1
```

Der Wert für LZ wird in jedem Fall erhöht, damit die DO-LOOP-Schleife in Zeile 50360 unter allen Bedingungen den (exakten) neuen Wert für LZ ermitteln kann.

- *Zeile 50650: Cursor-Links-Bewegung*

Dort mag die doppelte Verwendung von IF...THEN ein wenig verwirren. Diese ist aber nur eine optimierte Form für:

```
IF A$=CHR$(157) AND (NOT FE) THEN P=P-1:GOTO 50330
```

Die in Zeile 50650 stehende Version ist aber etwas schneller.

- *Zeile 50690: Cursor-Rechts-Bewegung*

Bei der Cursorbewegung nach rechts wird außer »IF A$=CHR$(29)« eine weitere IF...THEN-Abfrage, ob die Cursorbewegung erfolgen darf, dadurch vermieden, daß der logische Ausdruck

```
P<>MX AND P<>LZ+1
```

in Klammern gesetzt wird. Ist diese Bedingung wahr, so wird 1 addiert (die Klammer wird -1, das Vorzeichen »-« vor der Klammer führt zur Addition von 1), andernfalls ändert sich P nicht, da die Klammer dann 0 liefert und 0 bei der Subtraktion neutral ist.

- *Zeile 51010: Rücksprung zur Eingabeschleife*

Wenn der Rücksprung zur Eingabeschleife ab Zeile 50470 entsprechend verschoben wird (z.B. zur Zeile 52000), können Sie Platz für eigene Erweiterungen schaffen. Ich wüßte aber nicht, was dieser Eingaberoutine noch an Funktionen fehlt.

Anwendung der Routine

Die vorgestellte Routine ist zwar, wie wir soeben gesehen haben, schon an einigen Stellen optimiert, aber durch Entfernen aller überflüssigen Leerzeichen und REMs sowie dem Zusammenfassen mehrerer Befehle in eine Zeile können Speicherplatzbedarf und Arbeitsdauer noch drastisch verringert werden, was den Wert der Routine deutlich steigert. Deshalb sollte bei der

Weiterverwendung der Eingaberoutine in eigenen Programmen folgende Version verwendet werden, die zur alten Fassung parameter-kompatibel ist:
Beschreibung: Komprimierte Eingaberoutine ab Zeile 50000
Filename: »eingabe.komprim.«

```
50000 P=1:E1$=E$+ZR$+CHR$(17)+CHR$(19)+CHR$(20)+CHR$(29)+CHR$(145)+CHR$(147)+CHR
$(148)+CHR$(157)
50010 IF(NOTVB)THENIN$="":FORF=1TOMX:IN$=IN$+"_":NEXT
50020 IF VB THENLZ=MX:DOUNTILLEN(IN$)=MX:IN$=IN$+"_":LOOP:GOTO50040
50030 LZ=2
50040 FL=(P=MX):FE=(P=1)
50050 DOWHILEMID$(IN$,LZ,1)="_"ANDLZ>1:LZ=LZ-1:LOOP
50060 CHAR,S,Z,IN$
50070 CHAR,S+P-1,Z,MID$(IN$,P,1),1
50080 DO:GETKEY A$:LOOP UNTIL INSTR(E1$,A$)
50090 IFINSTR(E$,A$)THENBEGIN
50100 MID$(IN$,P,1)=A$
50110 P=P+1+(P=MX):LZ=LZ+1
50120 GOTO50040
50130 BEND
50140 IFINSTR(ZR$,A$)THENBEGIN
50150 IFLZ<MLTHENGOTO50080
50160 IN$=LEFT$(IN$,LZ):H$=IN$:IN$=""
50170 FORF=1TOLZ
50180 IFMID$(H$,F,1)<>"_"THENIN$=IN$+MID$(H$,F,1)
50190 NEXT:IFLEN(IN$)<MLTHENIN$=H$:GOTO50080
50200 RETURN
50210 BEND
50220 IFA$=CHR$(157)THENIFNOTFETHENP=P-1:GOTO50040
50230 IFA$=CHR$(147)THENIN$="":GOTO50000
50240 IFA$=CHR$(29)THENP=P-((P<>MX)AND(P<>LZ+1)):GOTO50040
50250 IFA$=CHR$(145)THENBEGIN
50260 IFP=1THEN50080
50270 H$=IN$:IN$=""
50280 FORF=PTOMX:IN$=IN$+MID$(H$,F,1):NEXT:LZ=LEN(IN$):DOUNTILLEN(IN$)=MX:IN$=IN
$+"_":LOOP:P=1
50290 GOTO 50040
50300 BEND
50310 IFA$=CHR$(17)THENBEGIN
50320 FORF=PTOMX:MID$(IN$,F,1)="_":NEXT:LZ=P
50330 GOTO50040
50340 BEND
50350 IFA$=CHR$(20)THENBEGIN
50360 IFFETHEN50080
50370 IN$=LEFT$(IN$,P-2)+RIGHT$(IN$,MX-P+1)
50380 DOUNTILLEN(IN$)=MX:IN$=IN$+"_":LOOP
50390 P=P-1
50400 GOTO50040
50410 BEND
50420 IFA$=CHR$(148)THENBEGIN
50430 IFLZ=MXTHEN50080
50440 IFFLTHEN50080
50450 IN$=LEFT$(IN$,P-1)+"_"+RIGHT$(IN$,MX-P+1):IN$=LEFT$(IN$,MX)
50460 LZ=LZ+1
50470 GOTO50040
50480 BEND
50490 IFA$=CHR$(19)THENP=1:GOTO50040
50500 GOTO50080
```

Zur Ermittlung der Parameter ist folgendes Demoprogramm äußerst hilfreich, das schon angesprochen wurde:
Beschreibung: Demonstrationsprogramm zur Eingaberoutine
Filename: »demo.eingabe«

```
100 REM *** DEMO ZUR EINGABE-ROUTINE ***
110 PRINT "EINGABE-PARAMETER:";CHR$(13)
120 S=5:INPUT "SPALTE";S
130 Z=3:INPUT "ZEILE";Z
140 ML=5:INPUT "MINDESTLAENGE";ML
150 MX=25:INPUT "MAXIMALLAENGE";MX
160 VB=0:INPUT "VORBELEGUNGSFLAG";VB
170 IF VB THEN INPUT "VORBELEGUNGSSTRING";IN$
180 E$="ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890":INPUT "ERLAUBTE ZEICHEN";E$
190 ZR$="":INPUT "ZEICHEN FUER EINGABE-ENDE";ZR$:ZR$=ZR$+CHR$(13):PRINT"(UND NAT
UERLICH 'RETURN')"
200 A$="N":INPUT "ALLES RICHTIG (J/N)";A$:IF A$<>"J" THEN RUN
210 SCNCLR:GOSUB 50000
220 PRINT:PRINT:PRINT "IHRE EINGABE:";IN$
230 PRINT "LAENGE DER EINGABE:";LEN(IN$);"ZEICHEN"
240 PRINT
250 GOTO 120
260 :
270 :
280 :
290 REM EINGABEROUTINE AB 50000
300 :
50000 P=1:E1$=E$+ZR$+CHR$(17)+CHR$(19)+CHR$(20)+CHR$(29)+CHR$(145)+CHR$(147)+CHR
$(148)+CHR$(157)
50010 IF(NOTVB)THENIN$="":FORF=1TOMX:IN$=IN$+"_":NEXT
50020 IF VB THENLZ=MX:DOUNTILLEN(IN$)=MX:IN$=IN$+"_":LOOP:GOTO50040
50030 LZ=2
50040 FL=(P=MX):FE=(P=1)
50050 DOWHILEMID$(IN$,LZ,1)="_"ANDLZ>1:LZ=LZ-1:LOOP
50060 CHAR,S,Z,IN$
50070 CHAR,S+P-1,Z,MID$(IN$,P,1),1
50080 DO:GETKEY A$:LOOP UNTIL INSTR(E1$,A$)
50090 IFINSTR(E$,A$)THENBEGIN
50100 MID$(IN$,P,1)=A$
50110 P=P+1+(P=MX):LZ=LZ+1
50120 GOTO50040
50130 BEND
50140 IFINSTR(ZR$,A$)THENBEGIN
50150 IFLZ<MLTHENGOTO50080
50160 IN$=LEFT$(IN$,LZ):H$=IN$:IN$=""
50170 FORF=1TOLZ
50180 IFMID$(H$,F,1)<>"_"THENIN$=IN$+MID$(H$,F,1)
50190 NEXT:IFLEN(IN$)<MLTHENIN$=H$:GOTO50080
50200 RETURN
50210 BEND
50220 IFA$=CHR$(157)THENIFNOTFETHENP=P-1:GOTO50040
50230 IFA$=CHR$(147)THENIN$="":GOTO50000
50240 IFA$=CHR$(29)THENP=P-((P<>MX)AND(P<>LZ+1)):GOTO50040
50250 IFA$=CHR$(145)THENBEGIN
50260 IFP=1THEN50080
50270 H$=IN$:IN$=""
50280 FORF=PTOMX:IN$=IN$+MID$(H$,F,1):NEXT:LZ=LEN(IN$):DOUNTILLEN(IN$)=MX:IN$=IN
$+"_":LOOP:P=1
50290 GOTO 50040
50300 BEND
50310 IFA$=CHR$(17)THENBEGIN
```

```
50320 FORF=PTOMX:MID$(IN$,F,1)="_":NEXT:LZ=P
50330 GOTO50040
50340 BEND
50350 IFA$=CHR$(20)THENBEGIN
50360 IFFETHEN50080
50370 IN$=LEFT$(IN$,P-2)+RIGHT$(IN$,MX-P+1)
50380 DOUNTILLEN(IN$)=MX:IN$=IN$+"_":LOOP
50390 P=P-1
50400 GOTO50040
50410 BEND
50420 IFA$=CHR$(148)THENBEGIN
50430 IFLZ=MXTHEN50080
50440 IFFLTHEN50080
50450 IN$=LEFT$(IN$,P-1)+"_"+RIGHT$(IN$,MX-P+1):IN$=LEFT$(IN$,MX)
50460 LZ=LZ+1
50470 GOTO50040
50480 BEND
50490 IFA$=CHR$(19)THENP=1:GOTO50040
50500 GOTO50080
```

Dieses Programm ist auch als Compilat (= Übersetzung eines Basic-Programms in ein Maschinenprogramm zum Zweck der Beschleunigung) mit dem Filenamen »c/demo.eingabe« auf der Programmdiskette abgespeichert, kann aber – wie alle compilierten Programme – nicht gelistet werden.
Aufgrund der höheren Geschwindigkeit sollte der compilierten Version zum Austesten bestimmter Parameter der Vorzug gegeben werden.
Die Bedeutung der Parameter haben wir schon am Anfang dieses Unterkapitels behandelt.
Nach folgendem Schema verwendet man die Eingaberoutine:
1. Parameter durch Variablendefinitionen setzen
2. Unterroutine mit »GOSUB 50000« aufrufen
3. In IN$ steht der eingegebene String
Nach dem Aufruf der Unterroutine verschwindet der Festcursor nicht; deshalb sollte das Eingabefeld unmittelbar nach der Eingabe gelöscht werden, was über

```
PRINT CHR$(27);"D";
```

möglich ist.

Ausbau der Routine zur Eingabemaske

Die vorgestellte Routine kann auch zur Programmierung einer Eingabemaske (was das ist, wurde am Anfang von 3.6.4 erklärt) verwendet werden. Dies zeigt folgendes Beispielprogramm:
Beschreibung: Eingabemaske mit Hilfe der Eingaberoutine
Filename: »eingabemaske«

```
100 REM *** EINGABEMASKE ***
110 ZR$=CHR$(13)+CHR$(27)+CHR$(141):VB=-1
120 DIM S(3),Z(3),MX(3),IN$(3),E$(3)
125 FOR I=0 TO 3:READ S(I),Z(I),MX(I),E$(I):NEXT
130 FD=0
140 S=S(FD):Z=Z(FD):MX=MX(FD):IN$=IN$(FD):E$=E$(FD)
150 GOSUB 50000:IN$(FD)=IN$
```

```
160 IF A$=CHR$(13) THEN FD=FD+1+4*(FD=3):GOTO 140
170 IF A$=CHR$(141)THEN FD=FD-1-4*(FD=0):GOTO 140
180 IF A$=CHR$(27) THEN BEGIN
190 :SCNCLR
200 :FOR F=0 TO 3
210 :PRINT IN$(F)
220 :NEXT
230 BEND
240 END
250 :
40000 DATA 5,5,20," ABCDEFGHIJKLMNOPQRSTUVWXYZ.-"
40010 DATA 5,7,20," ABCDEFGHIJKLMNOPQRSTUVWXYZ.-0123456789"
40020 DATA 5,9,20," ABCDEFGHIJKLMNOPQRSTUVWXYZ.-0123456789"
40030 DATA 5,11,15,"0123456789"
40040 :
40050 :
49999 END:REM *** AB 50000:EINGABE-UP
50000 P=1:E1$=E$+ZR$+CHR$(17)+CHR$(19)+CHR$(20)+CHR$(29)+CHR$(145)+CHR$(147)+CHR
$(148)+CHR$(157)
50010 IF(NOTVB)THENIN$="":FORF=1TOMX:IN$=IN$+"_":NEXT
50020 IF VB THENLZ=MX:DOUNTILLEN(IN$)=MX:IN$=IN$+"_":LOOP:GOTO50040
50030 LZ=2
50040 FL=(P=MX):FE=(P=1)
50050 DOWHILEMID$(IN$,LZ,1)="_"ANDLZ>1:LZ=LZ-1:LOOP
50060 CHAR,S,Z,IN$
50070 CHAR,S+P-1,Z,MID$(IN$,P,1),1
50080 DO:GETKEY A$:LOOP UNTIL INSTR(E1$,A$)
50090 IFINSTR(E$,A$)THENBEGIN
50100 MID$(IN$,P,1)=A$
50110 P=P+1+(P=MX):LZ=LZ+1
50120 GOTO50040
50130 BEND
50140 IFINSTR(ZR$,A$)THENBEGIN
50150 CHAR,S+P-1,Z,MID$(IN$,P,1)
50160 IN$=LEFT$(IN$,LZ):H$=IN$:IN$=""
50170 FORF=1TOLZ
50180 IFMID$(H$,F,1)<>"_"THENIN$=IN$+MID$(H$,F,1)
50190 NEXT
50200 RETURN
50210 BEND
50220 IFA$=CHR$(157)THENIFNOTFETHENP=P-1:GOTO50040
50230 IFA$=CHR$(147)THENIN$="":GOTO50000
50240 IFA$=CHR$(29)THENP=P-((P<>MX)AND(P<>LZ+1)):GOTO50040
50250 IFA$=CHR$(145)THENBEGIN
50260 IFP=1THEN50080
50270 H$=IN$:IN$=""
50280 FORF=PTOMX:IN$=IN$+MID$(H$,F,1):NEXT:LZ=LEN(IN$):DOUNTILLEN(IN$)=MX:IN$=IN
$+"_":LOOP:P=1
50290 GOTO 50040
50300 BEND
50310 IFA$=CHR$(17)THENBEGIN
50320 FORF=PTOMX:MID$(IN$,F,1)="_":NEXT:LZ=P
50330 GOTO50040
50340 BEND
50350 IFA$=CHR$(20)THENBEGIN
50360 IFFETHEN50080
50370 IN$=LEFT$(IN$,P-2)+RIGHT$(IN$,MX-P+1)
50380 DOUNTILLEN(IN$)=MX:IN$=IN$+"_":LOOP
50390 P=P-1
50400 GOTO50040
50410 BEND
50420 IFA$=CHR$(148)THENBEGIN
50430 IFLZ=MXTHEN50080
```

```
50440 IFFLTHEN50080
50450 IN$=LEFT$(IN$,P-1)+"_"+RIGHT$(IN$,MX-P+1):IN$=LEFT$(IN$,MX)
50460 LZ=LZ+1
50470 GOTO50040
50480 BEND
50490 IFA$=CHR$(19)THENP=1:GOTO50040
50500 GOTO50080
```

Wenn auch nicht alle Eingabefelder sofort sichtbar sind, so kann doch mit <RETURN> und <SHIFT>+<RETURN> zwischen diesen gewechselt werden. In den Zeilen 100-49999 befindet sich das Hauptprogramm, das die Eingabemaske steuert. In der Variablen FD steht immer die Nummer des aktuellen Eingabefeldes, von denen es 4 gibt (0-3).
Mit <ESC> wird die gesamte Eingabemaske bestätigt.
Hierfür wird ZR$ verwendet; dieser String enthält jetzt folgende Tastendrucke (in Klammern der CHR$-Code):

<RETURN>	(13)	um 1 Eingabefeld nach unten wechseln
<SHIFT>+<RETURN>	(141)	um 1 Eingabefeld nach oben wechseln
<ESC>	(27)	gesamte Eingabe beenden

Die Variablen S,Z,MX,IN$ und E$ für alle Felder werden in den gleichnamigen Arrays gespeichert; bei Bedarf werden die Werte daraus bezogen. Sie befinden sich übrigens in den DATA-Zeilen 40000-40030 und werden in Zeile 125 eingelesen.
Das Vorbelegungsflag VB muß immer auf -1 gesetzt sein, damit beim Verlassen eines Eingabefeldes nicht die Eingabe verlorengeht (bei erneutem Anwählen des entsprechenden Feldes muß sie nach wie vor vorhanden sein, sonst könnte zwischen den Eingabefeldern nicht gesprungen werden).
ML (Mindestlänge) muß auf 0 stehen; da das Setzen auf 0 bei RUN automatisch erfolgt, wird »ML=0« nicht zusätzlich angegeben. Eine Abfrage der Mindestlänge kann in die Behandlung von <ESC> (Zeilen 180-230) integriert werden.
Die Eingaberoutine steht ab Zeile 50000 und hat sich nur geringfügig geändert: In Zeile 50000 steht statt der Mindestlängenabfrage, die jetzt entfällt, der schon erwähnte Befehl zum Löschen des Festcursors. Dies ist wichtig, damit nicht in jedem Eingabefeld ein Cursor steht und dadurch Unklarheit entstünde, wo sich der »richtige« Cursor befindet.

3.6.5 Komfortable Menüs einfach programmiert

Viele Programme zeichnen sich dadurch aus, daß sie eine sehr anwenderfreundliche Menüsteuerung besitzen. Sieht man sich einige Menüs von Programmen kommerzieller Software-Hersteller an, wie zum Beispiel Textverarbeitungs- oder Dateiverwaltungsprogramme, so kann man hier mit den Cursortasten die einzelnen Programmfunktionen auswählen. Dabei werden mit den Cursortasten einzelne Menüpunkte hervorgehoben. Ist der gewünschte Menüpunkt erreicht, wird er mit einer Taste, z.B. <RETURN>, aktiviert.
Die einfachste Form der Menüsteuerung, die vor allem von Anfängern programmiert wird, ist

die Durchnumerierung der einzelnen Menüpunkte. Mit der Basic-Anweisung »ON...GOTO« können die Unterprogramme sehr schnell aufgerufen werden. Beispiel:

```
100 INPUT "MENUEPUNKT";A
110 ON A GOTO 1000,2000,3000,...,n
```

Etwas mehr Komfort bringt die Verwendung von GETKEY, die dem Anwender das Drücken von <RETURN> erspart:

```
100 PRINT "MENUEPUNKT?":GETKEY A$
110 ON VAL(A$) GOTO 1000,2000,3000,...,n
```

Eine solche Menüsteuerung wird in folgendem Programm angewandt:
Beschreibung: Diskettenhilfsprogramm mit einfacher Menüsteuerung
Filename: »menueprogramm 1«

```
10 REM ************************
20 REM *                      *
30 REM *  ANWENDUNGSBEISPIEL  *
40 REM *                      *
50 REM * ZUR EINFACHEN MENUE- *
60 REM *                      *
70 REM *   PROGRAMMIERUNG     *
80 REM *                      *
90 REM ************************
100 SCNCLR
110 PRINT"DISKETTENVERWALTUNGSPROGRAMM"
120 PRINT"============================"
130 CHAR,5, 5,CHR$(18)+" 1 "+CHR$(146)+" DIRECTORY"
140 CHAR,5, 7,CHR$(18)+" 2 "+CHR$(146)+" DISK-BEFEHL SENDEN"
150 CHAR,5, 9,CHR$(18)+" 3 "+CHR$(146)+" DISK-STATUS ABFRAGEN"
160 CHAR,5,11,CHR$(18)+" 4 "+CHR$(146)+" PROGRAMM BEENDEN"
170 CHAR,0,15,"BITTE DIE ENTSPRECHENDE TASTE DRUECKEN:"
180 DO:GETKEY A$:LOOP WHILE A$<"1" OR A$>"4":PRINT A$
190 :
200 ON VAL(A$) GOTO 1000,2000,3000,4000
210 :
1000 REM MENUEPUNKT 1 (DIRECTORY)
1010 SCNCLR
1020 DIRECTORY
1030 PRINT:CHAR,14,24,"TASTE DRUECKEN",1
1040 GETKEY A$
1050 RUN
1060 :
2000 REM MENUEPUNKT 2 (DISK-BEFEHL)
2010 PRINT:PRINT CHR$(27);"@";"DISK-BEFEHL:";
2020 POKE 208,1:POKE 842,34:REM ANFUEHRUNGSZEICHEN ALS ERSTES EINGABEZEICHEN, UM
 EINGABE DER TRENNZEICHEN (",″ ETC.) ZU ERLEICHTERN
2030 OPEN 1,0:INPUT#1,DB$:CLOSE 1:PRINT
2040 OPEN 1,8,15,DB$:CLOSE 1
2050 GOTO 130
2060 :
3000 REM MENUEPUNKT 3 (DISK-STATUS)
3010 PRINT:PRINT CHR$(27);"@";"DISK-STATUS:";DS$
3020 GOTO 130
3030 :
4000 REM MENUEPUNKT 4 (BEENDEN)
4010 PRINT:PRINT CHR$(27)+"@"
4020 END
```

Dieses einfache Diskettenhilfsprogramm ermöglicht die Anzeige des Disketteninhalts (Directory), das Senden eines Diskettenkommandos, die Abfrage des Diskettenstatus und das Programmende.
Über einen GETKEY-Befehl (Zeile 180) wird die Nummer des gewünschten Menüpunktes eingegeben, auf Richtigkeit der Eingabe wird mit »LOOP WHILE« geprüft (es sind nur <1>,<2>,<3> und <4> als Eingaben zugelassen) und in der Zeile 200 in den entsprechenden Programmteil gesprungen. Die Zahlen hinter ON...GOTO zeigen jeweils auf die erste Zeilennummer eines Programmteils. Zu den Zeilen 1000-4020 ist nichts weiter zu sagen, als daß dort die Routinen stehen, die zu den einzelnen Menüpunkten gehören.
Diese Form des Menüs, die in Listing 27 verwendet wird, ist allerdings noch recht unkomfortabel. Störend ist, daß man erst zu einer bestimmten Funktion die richtige Nummer eingeben muß. Und mehr als 9 oder (wenn ein Menüpunkt die Nummer 0 bekommt) maximal 10 Menüpunkte können nicht mehr mit Ziffern dargestellt werden, es müßte also auf Buchstaben oder den INPUT-Befehl (mit Eingabe zweistelliger Zahlen) ausgewichen werden.
Damit Sie gleich in den Genuß eines guten Menüs kommen, laden Sie das Programm »menueprogramm 2«, mit dem wir uns noch eine Weile beschäftigen werden, und starten Sie es. An der Bildschirmanzeige können Sie erkennen, daß die Menüpunkte intern mit den Zahlen 0-4 durchnumeriert werden, da diese Zahlen jeweils in Klammern hinter dem betreffenden Menüpunkt stehen.
Mit den Cursortasten können Sie das inverse Feld bewegen (<CRSR DOWN>= Inversfeld nach unten, <CRSR UP>= Inversfeld nach oben, <HOME>= Inversfeld in oberste Position). Mit <RETURN> kann ein Menüpunkt angewählt werden, mit <ESC> wird das Programm abgebrochen.
Wenn das Inversfeld, das sozusagen der Cursor des Menüs ist, mit <CRSR UP> über den obersten Punkt (Menüpunkt A) hinausbewegt wird, so kommt man zum untersten Menüpunkt (Menüpunkt E); ähnlich ist es, wenn mit <CRSR DOWN> das Inversfeld über den Menüpunkt E bewegt wird, was auf den obersten Punkt (Punkt A) führt. Dies zeigt auch Bild 3.4:

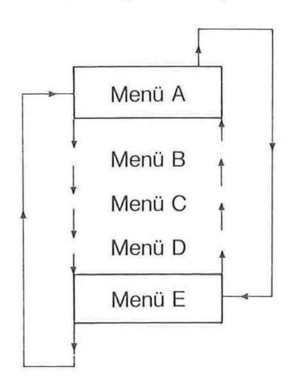


Bild 3.4: *Schemazeichnung (die Pfeile zeigen die Bewegungsrichtung des Inversfeldes an)*

Probieren Sie ohne weiteres alle Funktionen aus. Sollte Ihnen diese Art der Menüsteuerung gefallen, dann erfahren Sie im folgenden mehr über deren Aufbau.

Die Variablen von »Menüprogramm 2«

A enthält die Anzahl der Menüpunkte) (5). A ist eine Konstante, da sich der Wert nie ändert.

RT() ist ein Array, das für jeden Menüpunkt die Information enthält, ob er hervorgehoben werden soll. Ist RT(n)=1, so wird der Menüpunkt »n« invertiert, d.h. auf ihm liegt das Inversfeld. Andernfalls ist RT(n)=0.

P enthält die Nummer des derzeit angewählten Menüpunktes, d.h. des Menüpunktes, auf dem das Inversfeld liegt. P wird durch die Cursortasten geändert und wird zur Anwahl der Unterprogramme benötigt (460 ... THEN ON P+1 GOTO ...).

T$ enthält die in Zeile 450 geholte Taste und wird mehrfach über IF abgefragt, um die Taste auszuwerten.

Nachdem die Variablen beschrieben wurden, soll nun der Programmaufbau betrachtet werden.

Zeilen 100-280: Initialisierung

Zunächst werden die Variablen definiert (210-250), die Bildschirmfarben gesetzt (260) und im 80-Zeichen-Modus auf doppelte Geschwindigkeit geschaltet.
Bei den Variablendefinitionen wird P (Programmpunkt) auf Null gesetzt (Menüpunkt A), womit der oberste Menüpunkt als Ausgangsstellung gilt. Dieser Menüpunkt wird auch in Zeile 250 hervorgehoben (»RT(P)=1«). Die anderen Elemente von RT() sind nach Zeile 230 mit 0 belegt (0 bedeutet dabei »nicht revers«).
In Zeile 260 wird der 40-Zeichen-Bildschirm schwarz; der 80-Zeichen-Bildschirm ist in der Voreinstellung schwarz, weshalb keine zusätzliche Färbung erfolgt.

Zeilen 290-410: Menüpunkte ausgeben

Vor der Ausgabe jedes Menüpunktes wird eine individuelle Schriftfarbe gesetzt, da ein mehrfarbiges Menü übersichtlicher ist als ein einfarbiges.
Die Menüpunkt-Texte werden über CHAR ausgegeben, damit der letzte CHAR-Parameter (»,RT(.)«) über die Invertierung entscheidet. Die CHAR-Befehle stehen sehr schön übereinander, was dadurch erreicht wird, daß alle numerischen Parameter genau 2 Bildschirmpositionen einnehmen; bei einstelligen Zahlen wird ein Leerzeichen davor gesetzt. Dies ist nicht nur optisch schöner, sondern erleichtert auch das Editieren dieser Zeilen, da man sich an den unmittelbar darüber und/oder darunter stehenden Zeilen orientieren kann.
In Zeile 400 wird informationshalber der derzeit angewählte Menüpunkt angezeigt, was natürlich in der Endform eines Menüs wegfällt, da für den reinen Anwender die Funktionsweise des Programms relativ belanglos ist. Für uns, die wir aber solche Menüs selbst programmieren wollen, ist dies eine Verständnishilfe.

Zeilen 420-540: Tastaturabfrage

In diesem Programmteil wird die Tastatur abgefragt und das Inversfeld bewegt oder ein Menüpunkt ausgeführt. Dies geschieht, indem die Variable P (angewählter Menüpunkt) aktualisiert

wird (z.B. bei <CRSR DOWN> um 1 erhöht) und das Feld RT() entsprechende Werte bekommt (alle Elemente außer RT(P) müssen 0 sein).
Darauf gehen wir später noch einmal ein.

Zeilen 550-880: Routinen zu den Menüpunkten

Zu jedem Menüpunkt ist eine Routine vorhanden, die dann wieder zu Zeile 270 verzweigt.

Bewegung des Inversfeldes - ein simpler Trick

Die Struktur dieses Programms wäre damit, bis auf einige Besonderheiten, beschrieben. Um die Bewegung des Reversfeldes zu simulieren, muß das alte Bild bei jedem Tastendruck mit dem neuen Bild überschrieben werden. Die Frage ist: Wie erreicht man es, daß sich die Bilder absolut genau decken?
Die Antwort ist denkbar einfach: Da wir den CHAR-Befehl verwenden, werden die Texte, die sich ja im eigentlichen Sinn nicht ändern (nur die Reversdarstellung ändert sich), immer an denselben Bildschirmpositionen ausgegeben. Da das Inversfeld nur immer auf einem einzigen Menüpunkt liegt, werden die meisten Punkte nach wie vor in Normaldarstellung ausgegeben. Der Punkt, auf dem das Inversfeld vorher lag, wird jetzt aber deckungsgleich, jedoch nicht revers ausgedruckt, was den Eindruck entstehen läßt, das Inversfeld hätte sich von dort wegbewegt. Ähnlich ist es mit dem Punkt, auf den das Inversfeld gesteuert wurde: dieser wird jetzt revers ausgedruckt, und man meint, da sich der eigentliche Texte nicht verändert, das Inversfeld hätte sich auf diesen Punkt bewegt.
Dies wird Ihnen deutlicher, wenn Sie Zeile 310 ergänzen:

```
310 SCNCLR
```

Dann wird der Neuaufbau des Bildes sichtbar, weil der Bildschirm jedesmal komplett gelöscht wird. Dies macht sich dann durch ein Flackern des Bildes bemerkbar, aber diese Zeile 310 dient ja nur der Demonstration; in einem fertigen Programm steht diese Anweisung selbstverständlich nicht.

Die Steuerung des Inversfeldes

Wie wir wissen, hängt die Position des Inversfeldes davon ab, welche Werte im Array RT() und in der Variablen P stehen.
In Zeile 480 wird, bevor weitere IF-Abfragen durchgeführt werden, das Inversfeld an der aktuellen Position entfernt:

```
RT(P) = 0
```

In diesem Zustand würde kein einziger Menüpunkt revers dargestellt werden, was einem Fehlen des Inversfeldes entspricht.
Nach einigen IF-Behandlungen in den Zeilen 490-510 wird aber in Zeile 520 das Inversfeld gemäß der Variablen P gesetzt:

```
RT(P) = 1
```

Zwischen diesen beiden Stellen (Zeilen 480 und 520) wird ein neuer Wert für P errechnet, der sich aus der gewünschten Bewegung ergibt (<CRSR DOWN>, <CRSR DOWN> und <HOME> werden also in den Zeilen 490-510 behandelt).
Am einfachsten ist dies bei <HOME>, das durch CHR$(19) erreicht wird:

```
IF T$=CHR$(19) THEN P=0
```

Wenn <HOME> gedrückt wurde, so soll der oberste Menüpunkt angesprungen werden, der in der Numerierung die Zahl 0 hat. P wird dann auf 0 gesetzt, und in Zeile 520 wird dann RT(P), also RT(0), auf 1 gesetzt. An diesem Beispiel kann man den Ablauf gut verfolgen. Noch eine kurze Zusammenfassung:

```
480 RT(P)=0:REM Inversfeld zunächst löschen
...   IF-Abfragen, die bei <HOME> nicht zutreffen (Zeilen 490-510)
510 IF T$=CHR$(19) THEN P=0:REM Menüpunkt 0 setzen
520 RT(P)=1:REM Inversfeld auf Zielpunkt P (im Beispiel 0) setzen
530 GOTO 290:REM erneute Ausgabe der Menüpunkte, diesmal mit anderen Werten
                in P und RT()
```

Bei den Tasten <CRSR DOWN> und <CRSR UP> verläuft dies auf gleiche Weise, allerdings wird P auf andere Weise neu berechnet, da es sich schließlich auch um andere Tasten handelt. Bei <CRSR DOWN> wird Zeile 490 aktiv:

```
IF T$=CHR$(17) THEN P = P+1 + A*(P=(A-1))
```

Da die IF-Bedingung bei Drücken von <CRSR DOWN> erfüllt ist, können wir uns auf die Besprechung des THEN-Teils beschränken.

```
... P = P+1 + A*(P=(A-1))
```

Zunächst wird P um 1 erhöht, da die weiter unten am Bildschirm stehenden Menüpunkte höhere Kennziffern haben. Dafür ist »P= P+1« zuständig. Danach steht ein recht komplizierter Ausdruck, den wir mathematisch auflösen müssen:

```
... P = P+1 + A*(P=(A-1))
```

Da A eine Konstante ist, können wir den Zahlenwert 5 (Zeile 220) einsetzen:

```
... P = P+1 + 5*(P=(5-1))
```

Etwas weiter gerechnet, ergibt sich:

```
... P = P+1 + 5*(P=4)
```

Jetzt haben wir nur noch einen Klammerausdruck, der eine logische Abfrage ist. Wenn P=4 unwahr ist, also P nicht auf dem untersten Menüpunkt liegt, bekommt die Klammer den Wert 0:

```
bei P<>4: ... P = P+1 + 5*0
bei P<>4: ... P = P+1 + 0
bei P<>4: ... P = P+1
```

Dies deckt sich damit, daß das Inversfeld immer um 1 nach unten bewegt wird, wenn <CRSR DOWN> betätigt wird. Der Sonderfall P=4 (Inversfeld auf unterstem Menüpunkt) löst bekanntlich den Sprung zum obersten Menüpunkt (Punkt 0) aus. Auch dies wird von unserer Formel korrekt ausgeführt:

```
bei P=4: ... P = P+1 + 5*(-1)
bei P=4: ... P = P+1 + (-5)
bei P=4: ... P = P+1 - 5
bei P=4: ... P = P-4
```

Da P=4 Voraussetzung ist, können wir auf der rechten Seite der Zuweisung für P den Wert 4 einsetzen (den Fall, daß P nicht den Wert 4 hat, haben wir schon behandelt):

```
bei P=4: .. P = 4-4
bei P=4: ... P = 0
```

Fassen wir noch einmal zusammen:

```
Formel in Zeile 490:        P = P+1 + A*(P=(A-1))
nach Einsetzen von A:       P = P+1 + 5*(P=4)

wenn P=4 falsch ist:        P = P+1 + 5*0
                         => P = P+1

wenn P=4 wahr ist:          P = P+1 + 5*(-1)
                         => P = P-4
                         => P = 0
```

Wenn Zeile 490 mit IF konstruiert werden soll, müßte folgendes geschrieben werden:

```
490 IF T$=CHR$(17) THEN BEGIN
491 :IF P=4 THEN P=0:ELSE P=P+1
492 BEND
```

Dies ist zwar einsichtiger als unsere 1-Zeilen-Lösung, aber dafür auch etwas länger und langsamer. Ansonsten spricht aber nichts dagegen, daß Sie in Ihren Programmen mit einer solchen BEGIN-BEND-Konstruktion arbeiten, wenn Ihnen die lange Formel zu kompliziert erscheint. Folgendes ist aber nicht korrekt (siehe 3.4.2):

```
490 IF T$=CHR$(17) THEN IF P=4 THEN P=0:ELSE P=P+1
```

Betrachtet man Zeile 500, so sieht man, daß diese in ihrem Aufbau der Zeile 490, die wir soeben besprochen haben, sehr ähnelt.
Die Situation bei Betätigen von <CRSR UP> ist die, daß das Inversfeld um eine Position nach oben bewegt werden muß. Dazu ist nur P um 1 herunterzuzählen, da weiter oben stehende Menüpunkte niedrigere Kennziffern haben. Wird aber vom obersten Menüpunkt (Kennziffer 0) aus eine <CRSR UP>-Bewegung durchgeführt, soll der unterste Menüpunkt angesprungen werden.
Die Berechnung steht in Zeile 500, wobei uns wieder nur der THEN-Teil interessiert:

```
... P = P-1 - A*(P=0)
```

Hier noch das für beide Versionen gültige Flußdiagramm als Bild 3.5:

Anfang

If
T$="(Cursor up)"
nein
ja

P=P-1

If
P<0
nein
ja

P=Unterster Menüpunkt

Ende

Bild 3.5: *Bewegung des hervorgehobenen Feldes nach oben als Flußdiagramm*

Wir ersetzen die Konstante A (Anzahl der Menüpunkte) durch ihren in Zeile 220 zugewiesenen Wert 5:

```
... P = P-1 - 5*(P=0)
```

Jetzt sind wieder zwei Fälle zu betrachten:

```
P=0 ist wahr:          ... P = P-1 - 5*(-1)
                       ... P = P-1 - (-5)
                       ... P = P-1 + 5
                       ... P = P+4
```

Da in diesem Fall P den Wert 0 hat, kann P auf der rechten Seite der Zuweisung durch 0 ersetzt werden:

```
... P = 0+4
... P = 4
```

4 ist die Nummer des untersten Menüpunktes.

```
P=0 ist falsch:          ... P = P-1 - 5*0
                         ... P = P-1 - 0
                         ... P = P-1
```

Solange P=0 falsch ist und sich somit das Inversfeld nicht auf dem obersten Menüpunkt befindet, muß nur 1 subtrahiert werden.
Auch hier wollen wir die umständlich erscheinende Schreibweise durch eine IF...THEN-Konstruktion ersetzen, die dieselbe Wirkung hat, aber etwas langsamer und speicherplatzaufwendiger ist:

```
500 IF T$=CHR$(145) THEN BEGIN
501 :IF P=0 THEN P=4:ELSE P=P-1
502 BEND
```

Bei der Erklärung von IF...THEN...ELSE in Abschnitt 3.4.2 können Sie auch nachlesen, warum folgende Konstruktion nicht funktioniert:

```
500 IF T$=CHR$(145) THEN IF P=0 THEN P=4:ELSE P=P-1
```

Das Flußdiagramm der korrekten Lösung ist als Bild 3.6 zu sehen:

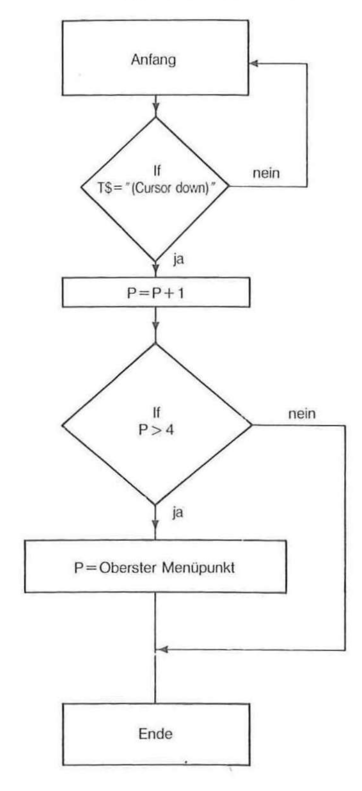


Bild 3.6: *Bewegung des hervorgehobenen Feldes nach unten als Flußdiagramm*

Ich nehme an, daß Ihnen das Prinzip jetzt klar geworden ist. Sollten Sie Schwierigkeiten mit den raffinierten Formeln haben, können Sie in Ihren Programmen auch die vorgestellten Lösungen mit Lösungen mit IF...THEN...ELSE verwenden, die schon mit Grundkenntnissen in Basic verstanden werden können.
Wir werden uns jetzt mit der Anwendung dieser Form der Menüsteuerung in der Praxis befassen und einige Änderungen bzw. Ergänzungen vornehmen. Schließlich wird noch eine Routine vorgestellt, zu deren Gebrauch überhaupt kein Verständis der Funktionsweise erforderlich ist.
Als erstes wollen wir das Menüprogramm optimieren, indem

- REM-Kommentare entfernt werden
- Zeilen komprimiert (zusammengefaßt) werden
- die Konstante A durch den Zahlenwert ersetzt wird

Dadurch wird das Programm kürzer und schneller:
Beschreibung: Optimierte Version von »Menüprogramm 2«
Filename: »menueprogramm 3«

```
100 REM "MENUEPROGRAMM 3" ( = GEKUERZTES "MENUEPROGRAMM 2")
110 CLR:DIM RT(4):P=0:RT(P)=1
120 COLOR 0,1:COLOR 4,1:COLOR 5,1:PRINTCHR$(142):SCNCLR
130 IF RWINDOW(2)=80 THEN FAST:REM FAST-MODUS BEI 80-ZEICHEN-DARSTELLUNG
140 COLOR 5,8:CHAR,3,0,"MENUESTEUERUNG - BEISPIELPROGRAMM"
150 CHAR,3,1,"=======(GEKUERZTE VERSION)======="
160 COLOR 5,3:CHAR,9, 6,"MENUEPUNKT A",RT(0):CHAR,23, 6,"(0)"
170 COLOR 5,4:CHAR,9, 8,"MENUEPUNKT B",RT(1):CHAR,23, 8,"(1)"
180 COLOR 5,5:CHAR,9,10,"MENUEPUNKT C",RT(2):CHAR,23,10,"(2)"
190 COLOR 5,6:CHAR,9,12,"MENUEPUNKT D",RT(3):CHAR,23,12,"(3)"
200 COLOR 5,7:CHAR,9,14,"MENUEPUNKT E",RT(4):CHAR,23,14,"(4)"
210 GETKEY T$
220 IF T$=CHR$(13) THEN ON P+1 GOTO 300,350,380,410,440
230 IF T$=CHR$(27) THEN PRINT:PRINT:PRINT:PRINT"PROGRAMMABBRUCH.":END
240 RT(P) = 0
250 IF T$=CHR$(17) THEN P=P+1+5*(P=4)
260 IF T$=CHR$(145)THEN P=P-1-5*(P=0)
270 IF T$=CHR$(19) THEN P=0
280 RT(P)=1
290 GOTO 140
300 SCNCLR:PRINT"SIE HABEN DEN OBERSTEN MENUEPUNKT ANGE-"
310 PRINT:PRINT"WAEHLT. DIESER HAT IN DER INTERNEN NU-"
320 PRINT:PRINT"MERIERUNG DES PROGRAMMS DIE NUMMER:"P
330 PRINT:PRINT:PRINT"BITTE DRUECKEN SIE EINE TASTE"
340 GETKEY A$:GOTO 120
350 SCNCLR:PRINT"SIE HABEN DEN ZWEITOBERSTEN MENUEPUNKT"
360 PRINT:PRINT"ANGEWAEHLT. INTERNE NUMMER: "P
370 GOTO 330
380 SCNCLR:PRINT"SIE HABEN DEN MITTLEREN MENUEPUNKT ANGE-"
390 PRINT:PRINT"WAEHLT. DESSEN INTERNE NUMMER IST:"P
400 GOTO 330
410 SCNCLR:PRINT"SIE HABEN DEN ZWEITUNTERSTEN MENUEPUNKT"
420 PRINT:PRINT"ANGEWAEHLT. INTERNE NUMMER:"P
430 GOTO 330
440 SCNCLR:PRINT"SIE HABEN DEN UNTERSTEN MENUEPUNKT ANGE-"
450 PRINT:PRINT"WAEHLT. INTERNE NUMMER:"P
460 GOTO 330
```

Jetzt belegt das Programm nur noch die Hälfte des vorher erforderlichen Speicherplatzes und hat auch eine höhere Verarbeitungsgeschwindigkeit; dabei ist dies nur eine teilweise Optimierung, denn es ließe sich ein noch besseres Ergebnis erzielen, wenn die genannten Optimierungsmethoden noch konsequenter durchgeführt würden. Ich habe mich aber für diesen Grad der Optimierung entschieden, damit noch eine möglichst große Ähnlichkeit zur ursprünglichen Version (Listing 28) besteht und Sie leichter vergleichen können.
Das Ersetzen der Variablen A durch den jeweiligen Zahlenwert erschwert zwar ein späteres Erweitern um weitere Menüpunkte oder das Streichen einer Option, aber als Beispiel sei ein Menü mit 4 (statt 5) Menüpunkten vorgestellt. Dabei handelt es sich um ein Diskettenverwaltungsprogramm wie »Menüprogramm 1«, aber mit einem komfortablen Menü. Die Programmabbruch-Möglichkeit über <ESC> ist dabei zwar entfernt worden, aber ansonsten ist die Steuerroutine (Zeilen 220-280) zum entsprechenden Programmabschnitt in Listing 29 (Zeilen 210-280) parallel aufgebaut:
Beschreibung: Diskettenverwaltungsprogramm mit komfortablem Menü
Filename: »menueprogramm 4«

```
100 REM DISKETTENVERWALTUNGSPROGRAMM
110 REM ** MIT KOMFORTABLEM MENUE **
120 :
130 CLR:DIM RT(3):P=0:RT(P)=1
140 COLOR 0,1:COLOR 4,1:COLOR 5,1:PRINTCHR$(142):SCNCLR
150 IF RWINDOW(2)=80 THEN FAST:REM FAST-MODUS BEI 80-ZEICHEN-DARSTELLUNG
160 COLOR 5,8:CHAR,4,0,"DISKETTENVERWALTUNGS - PROGRAMM"
170 CHAR,4,1,"==============================="
180 COLOR 5,3:CHAR,9, 6,"DIRECTORY ANZEIGEN",RT(0)
190 COLOR 5,4:CHAR,9, 8,"DISK-BEFEHL SENDEN",RT(1)
200 COLOR 5,5:CHAR,8,10,"DISK-STATUS ANZEIGEN",RT(2)
210 COLOR 5,6:CHAR,9,12,"PROGRAMM BEENDEN",RT(3)
220 GETKEY T$
230 IF T$=CHR$(13) THEN ON P+1 GOTO 300,340,370,400
240 RT(P) = 0
250 IF T$=CHR$(17) THEN P=P+1+4*(P=3)
260 IF T$=CHR$(145)THEN P=P-1-4*(P=0)
270 IF T$=CHR$(19) THEN P=0
280 RT(P)=1
290 GOTO 160
300 SCNCLR:DIRECTORY
310 CHAR,10,24,"<TASTE DRUECKEN>",1
320 GETKEY A$:SCNCLR
330 GOTO 160
340 PRINT CHR$(27);"@":PRINT:PRINT:PRINT"DISK-BEFEHL:";
350 POKE 208,1:POKE 842,34:OPEN 1,0:INPUT#1,A$:CLOSE 1:PRINT
360 OPEN 1,8,15,A$:CLOSE 1:GOTO 160
370 PRINT CHR$(27);"@":PRINT:PRINT
380 PRINT "DISK-STATUS:",DS$
390 GOTO 160
400 PRINT CHR$(27);"@":PRINT:PRINT
410 PRINT "PROGRAMMENDE."
420 END
```

Horizontales Menü

Sollen aus irgendwelchen Gründen die Menüpunkte horizontal angeordnet werden, ist auch dies möglich. Allerdings muß dann das Inversfeld nach links oder rechts bewegt werden, und <CRSR DOWN> bzw. <CRSR UP> wären dafür keine plausiblen Tasten. Am Algorithmus ändert sich nichts, aber die Funktion von <CRSR DOWN> wird bei horizontaler Anordnung von <CRSR RIGHT> und <CRSR UP> von <CRSR LEFT> übernommen.
Beschreibung: Horizontales Menü
Filename: »menueprogramm 4«

```
100 REM *** HORIZONTALES MENUE ***
110 REM *** (MENUEPROGRAMM  5) ***
120 :
130 CLR:DIM RT(4):P=0:RT(P)=1
140 COLOR 0,1:COLOR 4,1:COLOR 5,1:PRINTCHR$(142):SCNCLR
150 IF RWINDOW(2)=80 THEN FAST:REM FAST-MODUS BEI 80-ZEICHEN-DARSTELLUNG
160 COLOR 5,8:CHAR,3,0,"MENUESTEUERUNG - BEISPIELPROGRAMM"
170 CHAR,3,1,"=======(HORIZONTALES MENUE)======"
180 COLOR 5,3:CHAR, 0,20,"PUNKT A",RT(0)
190 COLOR 5,4:CHAR, 8,20,"PUNKT B",RT(1)
200 COLOR 5,5:CHAR,16,20,"PUNKT C",RT(2)
210 COLOR 5,6:CHAR,24,20,"PUNKT D",RT(3)
220 COLOR 5,7:CHAR,32,20,"PUNKT E",RT(4)
230 GETKEY T$
240 IF T$=CHR$(13) THEN COLOR 5,2:CHAR,0,5,"ANGEWAEHLTER MENUEPUNKT :"+STR$(P),1
:PRINT:END
250 IF T$=CHR$(27) THEN PRINT:PRINT:PRINT:PRINT"PROGRAMMABBRUCH.":END
260 RT(P) = 0
270 IF T$=CHR$(29) THEN P=P+1+5*(P=4)
280 IF T$=CHR$(157)THEN P=P-1-5*(P=0)
290 IF T$=CHR$(19) THEN P=0
300 RT(P)=1
310 GOTO 160
```

In Listing 31 wurde der größte Teil aus Listing 29 übernommen, aber die beiden ersten Parameter des CHAR-Befehls (Spalte, Zeile) sind dahingehend geändert worden, daß die Ausgabe horizontal ist.
Außerdem wurden die IF-Abfragen in den Zeilen 270 und 280 auf CHR$(29) für <CRSR RIGHT> anstelle von CHR$(17) für <CRSR DOWN> bzw. auf CHR$(157) für <CRSR LEFT> anstelle von CHR$(145) für <CRSR UP> geändert. Die letzte tiefgreifende Änderung ist, daß jetzt die Menüpunkte nicht mehr als Routinen über ON...GOTO aufgerufen werden, sondern - weil das Prinzip von ON...GOTO klar ist - nur die Nummer des angewählten Menüpunktes ausgegeben wird. Nach folgender Änderung wird nicht die Nummer, sondern der Buchstabe (A-E) ausgegeben, falls Ihnen dies besser gefällt:

```
240 IF T$=CHR$(13) THEN COLOR 5,2:CHAR,0,5,"ANGEWAEHLTER MENUE-
PUNKT : "+CHR$(65+P),1:PRINT:END
```

Der Code für den Buchstaben »A« ist die 65 (siehe C128-Handbuch, »Commodore-128-Codetabelle« ab Seite A-7). Alle im Alphabet folgenden Buchstaben haben jeweils eine um 1 größere Codenummer. Aufgrund dieser Tatsache läßt sich der Buchstabe des Menüpunktes mit der kleinen Rechnung »CHR$(65+P)« ermitteln.

Zweidimensionales Menü

Ein zweidimensionales Menü bringen Sie jetzt bitte bei eingelegter Programmdiskette mit RUN»MENUEPROGRAMM 6« auf den Bildschirm.
In zweidimensionalen Menüs werden alle vier Cursortasten miteinbezogen. <CRSR UP> und <CRSR DOWN> werden im Prinzip wie im vertikalen Menü zur Ansteuerung der vertikalen Ebenen (A-D,B-E,C-F) verwendet. <CRSR RIGHT> und <CRSR LEFT> schalten zwischen den horizontalen Ebenen (A-B-C,D-E-F) um.
Die entsprechenden Berechnungen stehen in den Zeilen 250-290. Diese müssen nicht wieder aufgeschlüsselt werden, da wir dies schon bei »Menüprogramm 2« getan haben. Deshalb gebe ich nur die Auflösung in BEGIN-BEND-Konstruktionen an, die Ihnen sicherlich mehr nützt (auch als Flußdiagramm in Bild 3.7):

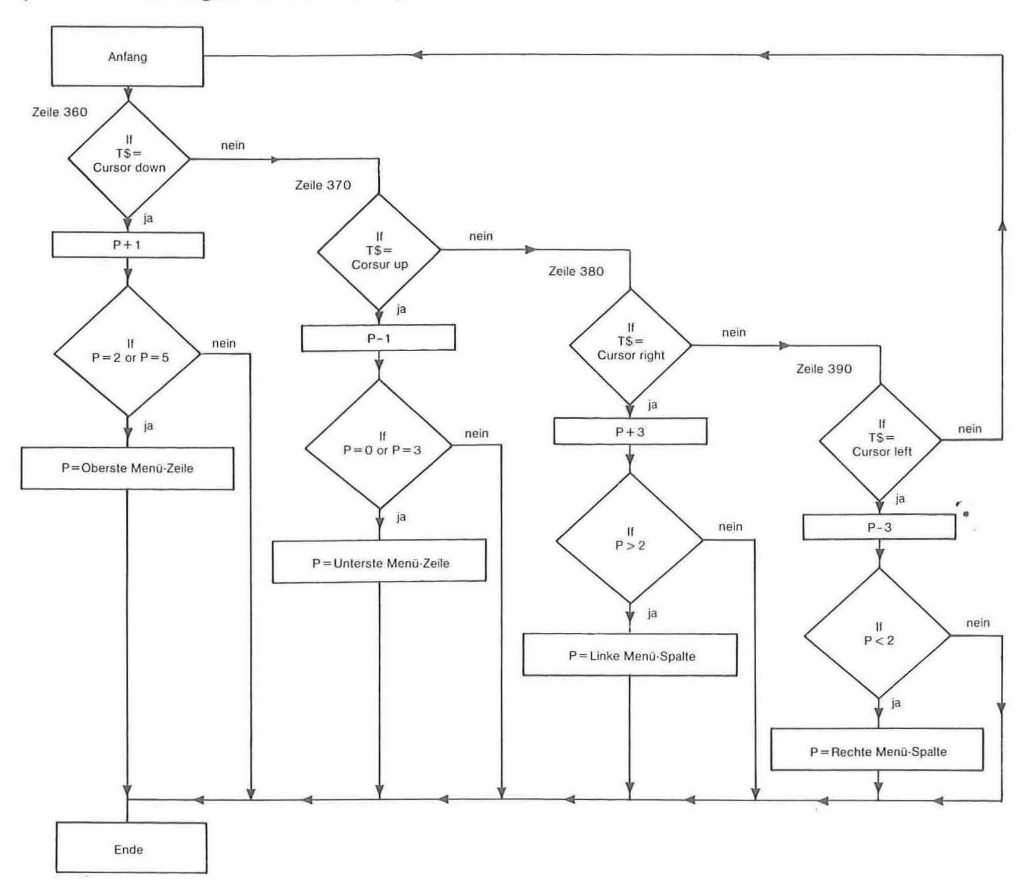


Bild 3.7: *Verschiebung des Inversfeldes im zweidimensionalen Menü*

```
250 IF T$=CHR$(17) THEN BEGIN:REM <CRSR DOWN>
251 :IF P=2 OR P=5 THEN P=P-2:ELSE P=P+1
252 BEND

260 IF T$=CHR$(145)THEN BEGIN:REM <CRSR UP>
261 :IF P=0 OR P=3 THEN P=P+2:ELSE P=P-1
262 BEND

270 IF T$=CHR$(29) THEN BEGIN:REM <CRSR RIGHT>
271 :IF P>2 THEN P=P-3:ELSE P=P+3
272 BEND

280 IF T$=CHR$(157)THEN BEGIN:REM <CRSR LEFT>
281 :IF P<3 THEN P=P+3:ELSE P=P-3
282 BEND
```

Ja-/Nein-Abfragen mit horizontalem Menü

Fast kein Programm kommt ohne eine Abfrage aus, die mit »JA« oder »NEIN« beantwortet werden muß. Natürlich kann diese Information über INPUT eingeholt werden, aber folgendes Programm hat eine bessere Lösung dieses Problems parat:
Beschreibung: Komfortable Ja-/Nein-Abfrage
Filename: »menueprogramm 7«

```
100 REM *** JA/NEIN-ABFRAGEN ***
110 REM *** MENUEPROGRAMM  7 ***
120 :
130 CLR:DIM RT(1):P=0:RT(P)=1
140 SCNCLR:PRINT"JA/NEIN-ABFRAGE"
150 PRINT"==============="
160 CHAR,0,5,"WAEHLEN SIE AUS (JA ODER NEIN):"
170 CHAR,32,5,"JA",RT(0)
180 CHAR,36,5,"NEIN",RT(1)
190 GETKEY T$
200 RT(P)=0
210 IF T$=CHR$(29) OR T$=CHR$(157) OR T$=" " THEN P=(NOT P) AND 1:REM "FLIPPEN"
DER VARIABLEN P
220 IF T$=CHR$(27) THEN PRINT:PRINT:PRINT"ABBRUCH.":END
230 IF T$<>CHR$(13) THEN RT(P)=1:GOTO 170
240 IF P=0 THEN PRINT:PRINT:PRINT"SIE HABEN SICH FUER 'JA' ENTSCHIEDEN."
250 IF P=1 THEN PRINT:PRINT:PRINT"IHRE ANTWORT LAUTET 'NEIN'."
```

Dabei wird mit <CRSR RIGHT>, <CRSR LEFT> oder der Leertaste zwischen »JA« und »NEIN« hin und her geschaltet. Da nur zwei Auswahlmöglichkeiten vorhanden sind (Ja/Nein), gibt es in der Wirkung keinen Unterschied zwischen einer Links- oder Rechts-Bewegung des Inversfeldes. Deshalb werden alle drei Tasten zu diesem Zweck mit einer einzigen Zeile behandelt (Zeile 210). Der THEN-Teil ändert den Wert P von 0 auf 1 bzw. von 1 auf 0. Dieses Wechseln nennt man »flippen« (umdrehen). Es wird durch NOT und AND realisiert:

```
... P=(NOT P) AND 1
```

Falls Sie einmal ein solches »Flippen« programmieren müssen, können Sie auf diese Formel zurückgreifen. Zwischen -1 und 0 bzw. 0 und -1 wird übrigens folgendermaßen gewechselt:

... P=NOT P

Da in unserem Beispielprogramm kein Menüpunkt die Nummer -1 hat, muß das Vorzeichen durch »AND 1« entfernt werden. Ebenso wäre die Basic-Funktion ABS dazu geeignet:

... P=ABS(NOT P) entspricht ... P=(NOT P) AND 1

Universelle Routine zur komfortablen Menüsteuerung

Die Programmierung der komfortablen Menüs ist zwar sehr reizvoll, aber zugleich auch mühsam. Deshalb sollte man, wenn mehrere Menüs in einem Programm vorkommen, eine Unterroutine einsetzen:
Beschreibung: Universelle Menüroutine
Filename: »menueprogramm 8«

```
100 REM *** ZWEIDIMENSIONALES MENUE ***
110 REM ***    (MENUEPROGRAMM 6)    ***
120 :
130 CLR:DIM RT(5):P=0:RT(P)=1
140 COLOR 0,1:COLOR 4,1:COLOR 5,1:PRINTCHR$(142):SCNCLR
150 IF RWINDOW(2)=80 THEN FAST:REM FAST-MODUS BEI 80-ZEICHEN-DARSTELLUNG
160 COLOR 5,8:CHAR,3,0,"MENUESTEUERUNG - BEISPIELPROGRAMM"
170 CHAR,3,1,"====(ZWEIDIMENSIONALES MENUE)===="
180 COLOR 5,3:CHAR, 5, 5,"PUNKT A",RT(0):COLOR 5,6:CHAR,20,5,"PUNKT D",RT(3)
190 COLOR 5,4:CHAR, 5, 7,"PUNKT B",RT(1):COLOR 5,7:CHAR,20,7,"PUNKT E",RT(4)
200 COLOR 5,5:CHAR, 5, 9,"PUNKT C",RT(2):COLOR 5,8:CHAR,20,9,"PUNKT F",RT(5)
210 GETKEY T$
220 IF T$=CHR$(13) THEN COLOR 5,2:CHAR,0,15,"ANGEWAEHLTER MENUEPUNKT :"+STR$(P),
1:PRINT:END
230 IF T$=CHR$(27) THEN PRINT:PRINT:PRINT:PRINT"PROGRAMMABBRUCH.":END
240 RT(P) = 0
250 IF T$=CHR$(17) THEN P=P+1+3*(P=2 OR P=5)
260 IF T$=CHR$(145)THEN P=P-1-3*(P=0 OR P=3)
270 IF T$=CHR$(29) THEN P=P+3+6*(P>2)
280 IF T$=CHR$(157)THEN P=P-3-6*(P<3)
290 IF T$=CHR$(19) THEN P=0
300 RT(P)=1
310 GOTO 160
```

Dieses Programm besteht aus einem Demonstrationsteil (Zeilen 10-40) und der Unterroutine (Zeilen 100-250). Die Menüroutine wird über GOSUB 100 aufgerufen, läuft sowohl im 40- als auch im 80-Zeichen-Modus und benötigt folgende Parameter:

– *Anzahl der Menüpunkte: Variable AM*

In AM muß die Anzahl der Menüpunkte stehen. Dabei darf man sich nicht davon beirren lassen, daß die Menüpunkte intern mit 0 beginnend numeriert werden. Zwei Menüpunkte ist die niedrigste sinnvolle Anzahl, nach oben besteht eigentlich keine Grenze außer der Bildschirmkapazität, da die Menüpunkte schließlich auch am Bildschirm durch einen Text vertreten sein wollen.

- *Flag für Initialisierung: Variable FI*

Wenn FI den Wert -1 hat, wird das Menü initialisiert, d.h. das Inversfeld wird auf den obersten Menüpunkt gesetzt. Beim ersten Aufruf eines Menüs ist dies dringend anzuraten. Andernfalls muß FI den Wert 0 bekommen.

- *Position eines Menüpunktes am Bildschirm: Array MP(,)*

Die Spalte eines Menüpunktes »n« am Bildschirm muß in MP(0,n) stehen, die Zeile in MP(1,n). Die Menüpunkte werden, wie bisher auch, mit 0 beginnend numeriert.
Es liegt in der Verantwortung der aufrufenden Routine, daß Zeile und Spalte sinnvoll gewählt sind (Menüpunkte mit höherer Nummer müssen weiter unten bzw. weiter rechts stehen).

- *Text eines Menüpunktes am Bildschirm: Array MP$()*

Der Text, der am Bildschirm für einen Menüpunkt »n« steht, ist in MP$(n) abzulegen. Dieser String darf auch Steuerzeichen, z.B. Farb-Steuerzeichen, beinhalten, aber bei Verwendung von RVS OFF (ASCII-Code: 146) muß man vorsichtig sein, damit die Invertierung des Textes am Bildschirm möglich ist.

- *Tasten zum Verlassen des Menüs: Variable AB$*

Außer <RETURN> können weitere Tasten definiert werden, die das Menü verlassen. Das Beispielprogramm läßt beispielsweise <ESC> zu (Zeile 10).
Wenn ein Programm viele hierarchische Menüs hat (Ober- und Unter-Menüs, ggf. noch Unter-Unter-Menüs usw.), müssen Tasten das Anspringen des Hauptmenüs, des unmittelbar in der Hierarchie über oder unter dem aktuellen Menü stehende anspringen. Diese Tasten legt man in AB$ ab, und nach dem Aufruf der Routine (»GOSUB 100«) kann das Hauptprogramm anhand von T$ feststellen, welche Taste das Verlassen des Menüs bewirkt hatte.

- *Horizontal-/Vertikal-Flag: Variable HV*

Ist HV=0, so handelt es sich um ein vertikales Menü, in dem die Tasten <CRSR DOWN> und <CRSR UP> die Menüpunkte ansteuern.
Ist HV=-1 (wie im Beispielprogramm, siehe Zeile 10), so handelt es sich um ein horizontales, über <CRSR RIGHT> und <CRSR LEFT> gesteuertes Menü.
Zweidimensionale Menüs sind nicht vorgesehen.

- *Nummer des angewählten Menüpunktes: Variable AP*

In AP steht nach dem GOSUB-Aufruf der Routine die Nummer des angewählten Menüpunktes. Wenn AP an die Routine übergeben werden soll, um die Position des Inversfeldes festzulegen, muß das Initialisierungsflag FI auf 0 stehen; außerdem muß das Feld RF(), welches die Revers-Informationen enthält, entsprechend gesetzt werden, was bei FI=-1 der Initialisierungsteil der Unterroutine übernimmt.

– *Variablendimensionierungen*

Wenn viele Menüpunkte möglich sind, müssen auch viele Informationen in den Arrays MP(,), MP$() und RF(), welches von der Unterroutine verwendet wird, stehen. Da die automatische Variablendimensionierung seitens des Basic-Interpreters nur Arrays mit bis zu 10 Elementen umfaßt, müssen die genannten Arrays bei mehreren Menüpunkten wie folgt dimensioniert werden (»n« muß durch durch die Anzahl der Menüpunkte ersetzt werden):

```
DIM MP(1,n-1),MP$(n-1),RF(n-1)
```

Dafür ist das Hauptprogramm selbst verantwortlich.
Ich wünsche Ihnen noch viel Spaß mit der universellen Routine, die Ihnen sicher hilft, einen Teil der Programmierarbeit einzusparen und sich mehr auf andere Programmteile zu konzentrieren.

3.6.6 Farbeinstellung über Menü

»Allen Menschen recht getan, ist eine Kunst, die keiner kann« – So oder ähnlich könnte man das Problem eines Programmierers beschreiben, der die Bildschirmfarben seines Werkes festlegen muß, denn der eine Anwender hat diesen Monitor, der andere jenen, und so kann es sein, daß der eine grüne Schrift auf schwarzem Grund am besten lesen kann, der andere Benutzer aus irgendwelchen Gründen aber gerade diese Farben nicht mag.
Abhilfe schafft hier nur die Einstellmöglichkeit durch eine Farbwahlroutine, die jedem seine individuelle Farbkombination gestattet, damit (hoffentlich) jeder zufrieden ist.
Folgende Farben stehen zur Wahl (Tabelle 3.14).

Tabelle 3.14: *Einzustellende Bildschirmfarben des C128*

40-Zeichen-Bildschirm		80-Zeichen-Bildschirm
	Zeichenfarbe	
Rahmenfarbe		Hintergrundfarbe
Hintergrundfarbe		

Da beim 80-Zeichen-Bildschirm nur zwei Farben eingestellt werden müssen, wollen wir für diesen etwas leichteren Fall zuerst eine Routine entwickeln, die wir dann für den 40-Zeichen-Bildschirm erweitern.
Im Prinzip handelt es sich um ein vertikales Menü, in dem aber nicht mit <RETURN> bestätigt wird, sondern über <+> und <-> eine über die Cursortasten angewählte Farbquelle geändert wird; diese Änderung soll sofort sichtbar sein.
In den Variablen H (Hintergrund) und V (Vordergrund, womit die Zeichen gemeint sind) stehen die entsprechenden Werte so, daß sie direkt über COLOR ausgeführt werden können.

Beschreibung: Farbwahlroutine für 80-Zeichen-Bildschirm
Filename: »farbwahl 80 z.«

```
100 REM   FARBEINSTELLUNG UEBER MENUE
110 REM  FUER DEN 80-ZEICHEN-BILDSCHIRM
120 :
130 CLR:DIM RT(1):AP=0:RT(AP)=1:H=1:V=8
135 SCNCLR:PRINT"BITTE STELLEN SIE NUN IHRE PERSOENLICHE FARBKOMBINATION EIN !"
140 COLOR 6,H:COLOR 5,V
150 CHAR,20,10,"HINTERGRUNDFARBE AENDERN (+ ODER -)",RT(0)
160 CHAR,20,12,"VORDERGRUNDFARBE AENDERN (+ ODER -)",RT(1)
170 :
180 GETKEY T$
190 IF T$=CHR$(17) OR T$=CHR$(145) THEN RT(AP)=0:AP=NOT AP AND 1:RT(AP)=1:GOTO 1
40
200 IF T$="+" THEN BEGIN
210 :IF AP=0 THEN H=H+1:IF H>16 THEN H=1
220 :IF AP=1 THEN V=V+1:IF V>16 THEN V=1
230 :GOTO 140
240 BEND
250 IF T$="-" THEN BEGIN
260 :IF AP=0 THEN H=H-1:IF H<1 THEN H=16
270 :IF AP=1 THEN V=V-1:IF V<1 THEN V=16
280 :GOTO 140
290 BEND
300 IF T$<>CHR$(13) THEN 180
```

Die Hauptschleife beginnt bei 140, wo zunächst die Farben gemäß den Variablen H und V gesetzt werden. Dann werden Tastendrücke behandelt. In AP steht, wie schon in der universellen Menüroutine, der aktuell angewählte Menüpunkt (im Beispielprogramm: 0 oder 1; da es sich wie bei der Ja-/Nein-Abfrage um zwei Zeichen handelt, wird AP wieder in Zeile 190 »geflippt«).
Dafür, daß die Werte für H und V im erlaubten Bereich (1-16) von COLOR liegen, sorgen IF-Abfragen in den Zeilen 210/220 und 260/270, die zwar auch durch Formeln mit logischen Ausdrücken ersetzt werden könnten, was aber nicht unbedingt nötig ist, da in diesem Fall dadurch keine große Zeit- oder Platzersparnis zu erzielen ist.
Mit <RETURN> wird die eingestellte Farbkombination angenommen und das Hauptprogramm, welches ab Zeile 301 stehen kann, gestartet.
Übrigens: Es kann bei der Farbwahl auch passieren, daß Schrift-und Hintergrundfarbe übereinstimmen, was wie ein leerer Bildschirm aussieht. Betätigen von <+> oder <-> schafft hier schnelle Abhilfe, da dann eine Farbe weiter geschaltet wird.
Eine vergleichbare Routine für den 40-Zeichen-Bildschirm ist folgende, die nur eine etwas aufwendigere Cursorsteuerung hat (jetzt stehen drei Menüpunkte zur Wahl):
Beschreibung: Farbwahlroutine für 40-Zeichen-Bildschirm
Filename: »farbwahl 40 z.«

```
100 REM   FARBEINSTELLUNG UEBER MENUE
110 REM  FUER DEN 40-ZEICHEN-BILDSCHIRM
120 :
130 CLR:DIM RT(2):AP=0:RT(AP)=1:H=1:R=7:V=8
140 SCNCLR:PRINT"BITTE STELLEN SIE NUN IHRE PERSOENLICHE FARBKOMBINATION EIN !"
150 COLOR 0,H:COLOR 4,R:COLOR 5,V
160 CHAR, 2,10,"HINTERGRUNDFARBE AENDERN (+ ODER -)",RT(0)
170 CHAR, 5,12,"RAHMENFARBE AENDERN (+ ODER -)",RT(1)
```

```
180 CHAR, 2,14,"VORDERGRUNDFARBE AENDERN (+ ODER -)",RT(2)
190 GETKEY T$
200 IF T$=CHR$(17) THEN RT(AP)=0:AP=AP+1+3*(AP=2):RT(AP)=1:GOTO 150
210 IF T$=CHR$(145)THEN RT(AP)=0:AP=AP-1-3*(AP=0):RT(AP)=1:GOTO 150
220 IF T$=CHR$(19) THEN RT(AP)=0:AP=0:RT(0)=1:GOTO 150
230 IF T$="+" THEN BEGIN
240 :IF AP=0 THEN H=H+1:IF H>16 THEN H=1
250 :IF AP=1 THEN R=R+1:IF R>16 THEN R=1
260 :IF AP=2 THEN V=V+1:IF V>16 THEN V=1
270 :GOTO 150
280 BEND
290 IF T$="-" THEN BEGIN
300 :IF AP=0 THEN H=H-1:IF H<1 THEN H=16
310 :IF AP=1 THEN R=R-1:IF R<1 THEN R=16
320 :IF AP=2 THEN V=V-1:IF V<1 THENV=16
330 :GOTO 150
340 BEND
350 IF T$<>CHR$(13) THEN 190
```

Hier darf das Hauptprogramm frühestens bei Zeile 351 beginnen, was sich natürlich durch ein RENUMBER ändert.

3.6.7 Windows – Fenster zum Bedienungskomfort

Der Begriff »Window« (Fenster) hat vor allem durch neuere Computer wie den AMIGA oder Atari ST einen größeren Bekanntheitsgrad erreicht, aber auch der C128 verfügt über eine, wenn auch sehr eingeschränkte, Window-Technik, die in 2.1.3 behandelt wurde. Wir wollen jetzt Windows im eigentlichen Sinn programmieren, die vom C128-Betriebssystem leider nicht unterstützt werden. Eine zentrale Funktion bekommt dabei der CHAR-Befehl, der zur PRINT-ähnlichen Ausgabe an beliebige Bildschirmpositionen dient.
Zunächst einmal wollen wir klären (bzw. wiederholen), was ein Window ist. Es handelt sich dabei um einen Bildschirmausschnitt, der als Teilbildschirm eingeblendet und separat, d.h. vom Hauptbildschirm unabhängig, behandelt wird. Die Ausgabe eines Windows geschieht durch Überschreiben eines Teils des Gesamtbildschirms mit dem Window-Text. Im Direktmodus ist dies von BASIC aus kaum möglich, da dies von einem Programm gesteuert werden muß; bei der Eingabe müssen Sie nach wie vor mit ESC-B und ESC-T arbeiten, was aber für Eingabezwecke in der Regel ausreicht.
Dadurch, daß der alte Bildschirm nicht gelöscht, sondern nur teilweise überlagert wird, ist der vorherige Bildschirminhalt noch zu einem beträchtlichen Teil erkennbar. Werden viele solche Überlagerungen durchgeführt, entsteht allerdings ein relativ chaotisches Aussehen des Bildschirms, da von einzelnen Windows nur noch Bruchstücke zu sehen sind. Grundvoraussetzung ist, daß das jeweils »aktuelle« Window in vollem Umfang eingeblendet ist.
Nach dieser kurzen Theorie wollen wir uns die Praxis ansehen.

Das erste Window-Programm

Wenn wir fürs erste keine größeren Ansprüche stellen, als daß ein als Window definierter Bereich unabhängig vom Gesamtbildschirm beschrieben wird, genügt uns folgendes Programm:

Beschreibung: Das erste Window-Programm
Filename: »windowprogramm 1«

```
100 REM ******************************
110 REM *                            *
120 REM * ANZEIGE EINER INFORMATION *
130 REM *                            *
140 REM *       IN EINEM WINDOW      *
150 REM *                            *
160 REM ******************************
170 REM *                            *
180 REM * START MIT ZUFAELLIGEM BE-  *
190 REM * SCHREIBEN DES BILDSCHIRMS  *
200 REM * ZUR HERVORHEBUNG DES WIN-  *
210 REM * DOWS:'RUN' ODER 'GOTO300'  *
220 REM *                            *
230 REM * START OHNE AENDERUNG DES   *
240 REM * BILDSCHIRMS: GOTO 360      *
250 REM *                            *
260 REM ******************************
270 :
280 :
290 :
300 REM BILDSCHIRM MIT ZUFAELLIGEN
310 REM ZEICHEN BESCHREIBEN:
320 :
330 SCNCLR
340 FOR I=1 TO 950:PRINT CHR$(RND(0)*48+48);:NEXT I:REM 950 ZUFAELLIGE ZEICHEN
350 :
360 REM WINDOW AUSGEBEN
370 REM ==============
380 :
390 CHAR,22, 4,"┌──────────────┐"
400 CHAR,22, 5,"|>> WINDOW <<|"
410 CHAR,22, 6,"├──────────────┤"
420 CHAR,22, 7,"|LINKE GR.:22|"
430 CHAR,22, 8,"|RECHTE G.:35|"
440 CHAR,22, 9,"|OBERE GR.: 4|"
450 CHAR,22,10,"|UNTERE G.:11|"
460 CHAR,22,11,"└──────────────┘"
470 :
480 GETKEY A$
```

Dieses Programm läuft im ASCII-Zeichensatz sowohl im 40- als auch im 80-Zeichen-Modus, ist allerdings größenmäßig für den 40er-Bildschirm zugeschnitten. Ein Beispiel für den 80-Zeichen-Bildschirm, das dessen größere Kapazität nutzt, kommt noch.
Wenn Sie das Programm ganz normal über RUN starten, wird zunächst der Bildschirm mit zufälligen Zeichen beschrieben. Dies erleichtert die Demonstration, daß nur der Bildschirmbereich des Windows genutzt wird und dieser eigenständig ist, denn die zufälligen Zeichen werden nur teilweise vom Window überschrieben.
Dann wird das Window selbst ausgegeben. Mit einem Tastendruck wird das Programm beendet.
In diesem Anwendungsbeispiel wird das Window rein demonstrativ genutzt, es zeigt lediglich seine »persönlichen« Daten (linke, rechte, obere und untere Grenze als Spalte oder Zeile) an. Rein von der Betrachtung des laufenden Programms kann man auch sehen, daß das Window optisch durch eine Umrahmung hervorgehoben wird. Diese Eingrenzung ist bei Commodore-

Computern dank der Grafikzeichen einfach realisierbar, denn Kasten- und Rahmen-Symbole sind in ausreichender Menge vorhanden. Im DIN-Zeichensatz haben diese zwar andere ASCII-Codes, sind aber fast über dieselben Tasten erreichbar. Deshalb müßte das Beispielprogramm für den DIN-Zeichensatz erst angepaßt werden, wobei zu beachten ist, daß der Querstrich jetzt nicht mehr auf <SHIFT>+<*>, sondern auf <CBM>+<*> liegt.
Die grafische Hervorhebung ist von elementarer Bedeutung, damit man klar ersehen kann, wo ein Window beginnt und wo es endet, denn Windows sind keine Spielerei, sondern sollen vielmehr eine ökonomische Nutzung des Bildschirms ermöglichen.
Bei einem Start des Programms über GOTO 360 wird das Überschreiben des Bildschirms durch Zufallszeichen übersprungen. Sie könnten zum Beispiel das Programm über LIST auf dem Bildschirm ausgeben und dann ohne vorheriges Löschen des Bildschirms starten. In jedem Fall wird das Window eingeblendet, egal, was vorher auf dem Bildschirm stand. Genau diese Eigenschaft macht die Flexibilität der Windows aus.

Die Programmiertechnik

Bis jetzt haben wir uns sehr wenig mit der Technik beschäftigt und das Ganze eher aus Anwendersicht betrachtet. Sie werden aber sehen, daß man keine besonderen Programmierkenntnisse braucht, um Windows zu realisieren. Im Prinzip ist es nur eine Frage der Planung des Programms und im besonderen der Bildschirmausgabe, die bei der Window-Programmierung wohl überlegt sein muß. Bei neueren Computern (AMIGA und Atari ST wurden bereits genannt) wird eine ausgereifte Window-Technik bereits vom Betriebssystem unterstützt; deswegen brauchen wir aber noch lange nicht zu verzweifeln, denn vieles, was die Großen können, schaffen die etwas Kleineren auch.
In diesem Zusammenhang sei noch einmal erwähnt, daß die sogenannte »Window-Technik« des C128 für unsere Zwecke unbrauchbar ist, denn es handelt sich nur um eine Verkleinerung des Bildschirms, aber schon mehrere Windows oder das Retten des unter einem Window liegenden Textes sind nicht mehr möglich.
Kommen wir zurück zur Programmierung. Weil die Ausgabe-Routinen des Computers nicht in der Lage sind, für uns darauf zu achten, daß wir nicht über die Grenzen eines Windows hinausschreiben, müssen wir solche Vorkehrungen selbst treffen. Als erstes müssen wir zur Ausgabe eines Windows die richtige Positionierung des Cursors beachten. Deshalb sollte man, wie unser erstes Window-Beispielprogramm, nur mit dem CHAR-Befehl arbeiten, der eine Angabe der Bildschirmposition von seiner Syntax her erfordert und uns sozusagen zwingt, dies zu beachten.
Beispiele hierzu können Sie den Zeilen 390-460 aus unserem ersten Programm entnehmen. Beim CHAR-Befehl ist das Schreiben der Window-Zeilen leichter, wenn man darauf achtet, daß der Ausgabestring jeweils in der gleichen Spalte am Bildschirm beim Editieren beginnt. Dann bekommt man eine genauere Vorstellung, wie der Text später am Bildschirm aussieht. Da die CHAR-Parameter »Spalte« und »Zeile« ein oder zwei Zeichen Länge haben, könnte es leicht sein, daß beim Listen die Ausgabestrings nicht genau, sondern leicht verschoben untereinander stehen. Dies kann man verhindern, indem man bei einstelligen Parametern vor den Parameter (aber hinter dem davor stehenden Komma) ein Leerzeichen setzt: In Zeile 390 wird

vor die einstellige Zahl 4 ein Leerzeichen gesetzt, damit die Zeile gleich lang ist wie zum Beispiel Zeile 450, in der der Parameter »Zeile« schon zweistellig ist. Dieses Einfügen von Leerzeichen ist für die Funktion der Zeile nicht erforderlich, denn Leerzeichen außerhalb von Anführungszeichen überliest der Basic-Interpreter bekanntlich; es ist aber dennoch kein übersteigertes Schönheitsbewußtsein, sondern - wie gesagt - eine große Hilfe beim Schreiben des Ausgabestrings.
Die Window-Ausgabe-Zeilen mit dem CHAR-Befehl müssen immer dieselbe Zahl als Spaltenangabe aufweisen; die Zeilenangabe wird von Zeile zu Zeile um 1 erhöht (siehe Zeilen 390-460), denn sonst würden wir immer nur dieselbe Window-Zeile drucken.
Wenn man nun darauf achtet, daß die Spaltenangabe immer gleich ist, die Zeilenangabe aber immer um 1 erhöht wird und nicht größer als die untere Window-Grenze wird, so hat man schon die linke und untere Grenze eingehalten. Die obere Grenze wird berücksichtigt, indem die erste Window-Ausgabe-Zeile in der obersten Window-Zeile beginnt. Auch die rechte Grenze kann schließlich leicht eingehalten werden, wenn die Ausgabestrings immer gleich sind und nicht länger als die Window-Breite sind. »Spalte + Stringlänge« darf die rechte Window-Grenze nicht überschreiten.
Wie Sie sehen, ist die Programmierung eines Windows ein Kinderspiel, solange man vorsichtig genug bei der Programmierung der Ausgabezeilen vorgeht und die Window-Grenzen einhält.

Windows am 80-Zeichen-Bildschirm

Da ein Window, wie jeder andere Text auch, Platz auf dem Bildschirm beansprucht, kann man auf einem größerem Bildschirm mehr oder größere Windows unterbringen als auf einem kleinen. Deshalb eignet sich der 80-Zeichen-Bildschirm besonders, denn er faßt doppelt soviele Zeichen wie der 40er-Bildschirm. Wir wollen uns diese größere Kapazität zunutze machen und lassen ein wesentlich platzaufwendigeres Window ausgeben:
Beschreibung: Window am 80-Zeichen-Bildschirm
Filename: »windowprogramm 2«

```
100 REM ******************************
110 REM *                            *
120 REM * ANZEIGE EINER INFORMATION *
130 REM *                            *
140 REM *       IN EINEM WINDOW      *
150 REM *                            *
160 REM ******************************
170 REM *                            *
180 REM * START MIT ZUFAELLIGEM BE-  *
190 REM * SCHREIBEN DES BILDSCHIRMS *
200 REM * ZUR HERVORHEBUNG DES WIN-  *
210 REM * DOWS:'RUN' ODER 'GOTO300' *
220 REM *                            *
230 REM * START OHNE AENDERUNG DES   *
240 REM * BILDSCHIRMS: GOTO 360      *
250 REM *                            *
260 REM * NUR IM 80-ZEICHEN-MODUS ! *
270 REM *                            *
280 REM ******************************
290 :
300 REM BILDSCHIRM MIT ZUFAELLIGEN
310 REM ZEICHEN BESCHREIBEN:
```

```
320 GRAPHIC 5:FAST:REM 80-ZEICHEN-MODUS UND FAST-MODUS EINSCHALTEN
330 SCNCLR
340 FOR I=1 TO 1999:PRINT CHR$(RND(0)*48+48);:NEXT I:REM 1999 ZUFAELLIGE ZEICHEN
350 :
360 REM WINDOW AUSGEBEN
370 REM ==============
380 :
460 CHAR ,22, 4," ┌──────────────────────────────────┐"
470 CHAR ,22, 5," │>>>> ETWAS GROESSERES WINDOW <<<<│"
480 CHAR ,22, 6," ├────────────────┬─────────────────┤"
490 CHAR ,22, 7," │LINKE GRENZE:22 │RECHTE GRENZE:56 │"
500 CHAR ,22, 8," │OBERE GRENZE: 4 │UNTERE GRENZE:16 │"
510 CHAR ,22, 9," ├────────────────┴─────────────────┤"
520 CHAR ,22,10," │DA DER 80-ZEICHEN-BILDSCHIRM DOP- │"
530 CHAR ,22,11," │PELT SOVIELE ZEICHEN AUFNEHMEN    │"
540 CHAR ,22,12," │KANN WIE DER 40'ER, SIND AUCH     │"
550 CHAR ,22,13," │SOLCH GROSSE WINDOWS, DIE SONST   │"
560 CHAR ,22,14," │ZUVIEL PLATZ AM BILDSCHIRM BEAN-  │"
570 CHAR ,22,15," │SPRUCHEN WUERDEN, MOEGLICH.       │"
580 CHAR ,22,16," └──────────────────────────────────┘"
590 :
600 GETKEY A$
```

Dieses Programm stützt sich wieder auf den CHAR-Befehl. In Zeile 320 werden 80-Zeichen-Bildschirm und doppelte Geschwindigkeit eingestellt. In den Zeilen 330 und 340 wird der Bildschirm mit zufälligen Zeichen beschrieben, was durch GOTO 360 zu umgehen ist, sofern der 80-Zeichen-Modus angewählt wurde. Ein beliebiger Tastendruck beendet das Programm.

Mehrere Windows

Bislang haben wir jeweils ein einziges Window ausgegeben. Folgendes Programm ist nun ein Anwendungsbeispiel, das sich mehrerer Windows bedient. Es handelt sich um ein Menü, wie es vor einem fiktiven Spielprogramm stehen könnte. Durch Drücken der Tasten <1>, <2> und <3> wird der Wert für Geschwindigkeit, Anzahl der Spieler oder Spielstufe erhöht; wird der höchste erlaubte Wert überschritten, so wird wieder 1 eingestellt.
<4> beendet das Programm.
Anhand dieses Beispiels kann man eine Reihe von Möglichkeiten der optischen Hervorhebung eines Windows erkennen (Kasten- und Rahmen-Symbole, Reversschrift, Sternchen usw.).
Bei mehreren Windows muß zusätzlich darauf geachtet werden, daß sich die Windows nicht gegenseitig überschreiben, was manchmal viel Tüftelei verursacht. Ansonsten erklärt sich das Programm durch die REM-Kommentare von selbst.

Text unter Window retten

Einen großen Nachteil haben alle bisher von uns ausgegebenen Windows gehabt: Der Text, der vorher an der Position des Windows stand, ist verlorengegangen, weil er durch das Window überschrieben wurde.
Dies macht in 50 Prozent aller Fälle nichts aus, aber was ist mit den anderen 50 Prozent?
Wir wollen den unter dem Window liegenden Bildschirmausschnitt retten, indem wir ihn über PEEK auslesen und in einem Variablenarray speichern. Wenn das Window verschwinden und der alte Text eingeblendet sein soll, wird das Array wieder in den Bildschirmspeicher geschrie-

ben. Diese Anforderung erfüllt folgendes Programm für den 40-Zeichen-Modus, das Sie bitte zuerst ausprobieren, bevor wir es dann besprechen:
Beschreibung: Window mit Textsicherung für den 40-Zeichen-Modus
Filename: »windowprogramm 4«

```
100 REM ********************************
110 REM *                              *
120 REM *    ANZEIGE EINES WINDOWS     *
130 REM *                              *
140 REM * UNTER BERUECKSICHTIGUNG DES *
150 REM *                              *
160 REM * UNTER DEM WINDOW LIEGENDEN   *
170 REM *                              *
180 REM *      BILDSCHIRMINHALTES      *
190 REM *                              *
200 REM ********************************
210 REM *                              *
220 REM *  NUR IM 40-ZEICHEN-MODUS !   *
230 REM *                              *
240 REM ********************************
250 :
260 B = 1024:REM BASISADRESSE DES BILDSCHIRMSPEICHERS IM 40-ZEICHEN-MODUS
270 DIM T(111):REM ARRAY ZUM RETTEN DES TEXTES (S. 350-)
280 :
290 GETKEY A$
300 IF A$="N" THEN 350:REM BEI "N" BILDSCHIRM NICHT ZUFAELLIG BESCHREIBEN
310 :
320 SCNCLR
330 FOR I=1 TO 999:PRINT CHR$(35+80*RND(0));:NEXT
340 :
350 REM AN WINDOW-POSITIONEN LIEGENDEN
360 REM TEXT RETTEN
370 :
380 REM DAS WINDOW BEANSPRUCHT:
390 REM DIE SPALTEN 23-36
400 REM I.D. ZEILEN  5-12
410 :
420 I=0:REM MIT INDEX 0 BEGINNEN
430 FOR Z= 4 TO 11:REM ZEILEN : 5-12
440 FOR S=22 TO 35:REM SPALTEN:23-36
450 T(I)=PEEK(B+40*Z+S):I=I+1:REM WERT EINLESEN, INDEX ERHOEHEN
460 NEXT S,Z
470 :
480 REM NUN WIRD DAS WINDOW AUSGEGEBEN:
490 :
500 CHAR,22, 4,"┌────────────┐"
510 CHAR,22, 5,"|>> WINDOW <<|"
520 CHAR,22, 6,"├────────────┤"
530 CHAR,22, 7,"|LINKE GR.:22|"
540 CHAR,22, 8,"|RECHTE G.:35|"
550 CHAR,22, 9,"|OBERE GR.: 4|"
560 CHAR,22,10,"|UNTERE G.:11|"
570 CHAR,22,11,"└────────────┘"
580 :
590 GETKEY A$
600 :
610 REM NUN WIRD DER TEXT UNTER DEM
620 REM WINDOW WIEDER ANGEZEIGT:
630 :
640 I=0
650 FOR Z= 4 TO 11
```

```
660 FOR S=22 TO 35
670 POKE B+40*Z+S,T(I):I=I+1:REM WERT SCHREIBEN, INDEX ERHOEHEN
680 NEXT S,Z
690 :
700 GETKEY A$
710 :
720 GOTO 350:REM WIEDER RETTEN UND WINDOW ANZEIGEN
```

Dieses Programm überschreibt auf Drücken einer Taste zunächst den Bildschirm mit zufälligen Zeichen, sofern nicht <N> gedrückt wurde. Dann wird ein Window ausgegeben und auf einen Tastendruck gewartet, auf den hin der Text, der ursprünglich unter dem Window lag, wieder eingeblendet wird. Erneuter Tastendruck gibt wieder das Window aus und so weiter.
Das Window kennen Sie schon aus »Windowprogramm 1«.
Befassen wir uns nun mit der Programmierung. In Zeile 260 wird die Basisadresse des Bildschirmspeichers festgelegt (1024). Zeile 270 dimensioniert ein 111+1 = 112 Elemente großes Array T(). Das »+1« ergibt sich daraus, das die Zählung (wie auch bei CHAR) mit 0 beginnt. Zwischen 0 und 111 liegen 112 Elemente T(0)...T(111).
In diesem Array werden später die »unter dem Window liegenden« Zeichen gespeichert. Die Größe ergibt sich aus der Anzahl der Spalten des Windows multipliziert mit der Anzahl der Window-Zeilen: 14*8=112.
Der an der Window-Position liegende Text muß noch vor der Ausgabe des Windows gerettet werden, da er ja durch das Window überschrieben wird. Dies erreichen wir in den Zeilen 420-460. Wie Sie sehen, wird auch hier – wie bei CHAR – von 0 an numeriert, in den REM-Kommentaren stehen die entsprechenden (um 1 erhöhten) Werte der anderen Zählweise, die mit 1 beginnt.
In Zeile 450 wird der Wert ins Array T() eingelesen und der Index innerhalb des Arrays (Variable I) erhöht.
Die Window-Ausgabe (Zeilen 480-570) bedarf keiner weiteren Erläuterung. In den Zeilen 640-680 wird dann das Array T() wieder in den Bildschirmspeicher geschrieben. An der Schleife ändert sich gegenüber 420-460 nur, daß jetzt der Wert nicht in T() geholt, sondern aus T() in den Bildschirmspeicher gepoket wird. Zum Schluß der Besprechung von »Windowprogramm 4« noch ein Hinweis: Dieses Programm läuft nur, wenn nicht gerade »BANK1« eingestellt wurde. Ganz Vorsichtige können den Befehl BANK0 oder BANK15 am Programmanfang anfügen.
Etwas schwieriger ist die »Rettungsaktion« beim 80-Zeichen-Bildschirm, da dessen Bildschirmspeicher nicht einfach über PEEK und POKE angesprochen werden kann, sondern indirekt adressiert wird. Folgendes Programm für den 80-Zeichen-Modus stellt das Window aus »Windowprogramm 2« dar, entspricht aber von der Bedienung her unserem letzten Beispiel.
Als kleine Ergänzung wird jetzt immer, wenn Sie eine Eingabe tätigen müssen, der Signalton über PRINT CHR$(7) erzeugt, da die 80-Zeichen-Version recht langsam ist und somit ein Signalton die Anwendung erleichtert.

Beschreibung: Window mit Text-Rettung für den 80-Zeichen-Modus
Filename: »windowprogramm 5«

```
100 REM ******************************
110 REM *                            *
120 REM * ANZEIGE EINER INFORMATION *
130 REM *                            *
140 REM *       IN EINEM WINDOW      *
150 REM *                            *
160 REM * MIT RETTUNG DES UNTER DEM *
170 REM *                            *
180 REM *     WINDOW  LIEGENDEN      *
190 REM *                            *
200 REM *     BILDSCHIRMINHALTES     *
210 REM *                            *
220 REM ******************************
230 REM *                            *
240 REM * NUR FUER 80-Z.-BILDSCHIRM *
250 REM *                            *
260 REM ******************************
270 :
280 DIM T(35*13-1):REM ARRAY ZUM RETTEN DES TEXTES "UNTER DEM WINDOW"
290 BANK 15
300 :
310 REM BILDSCHIRM MIT ZUFAELLIGEN
320 REM ZEICHEN BESCHREIBEN:
330 GRAPHIC 5:FAST:REM 80-ZEICHEN-MODUS UND FAST-MODUS EINSCHALTEN
340 PRINTCHR$(7);:GETKEY A$:IF A$="N" THEN 370
350 SCNCLR
360 FOR I=1 TO 1999:PRINT CHR$(RND(0)*48+48);:NEXT I:REM 1999 ZUFAELLIGE ZEICHEN
370 :
380 REM TEXT UNTER WINDOW RETTEN
390 REM =======================
400 I=0
410 FOR Z= 4 TO 16
420 FOR S= 22 TO 56
430 AD=Z*80+S
440 SYS DEC("CDCC"),AD/256,18
450 SYS DEC("CDCC"),AD AND 255,19
460 SYS DEC("CDDA"),,31:RREG T(I)
470 I=I+1
480 NEXT S,Z
490 :
500 CHAR ,22, 4,"┌─────────────────────────────────┐"
510 CHAR ,22, 5,"|>>>> ETWAS GROESSERES WINDOW <<<<|"
520 CHAR ,22, 6,"├───────────────┬─────────────────┤"
530 CHAR ,22, 7,"|LINKE GRENZE:22|RECHTE GRENZE:56 |"
540 CHAR ,22, 8,"|OBERE GRENZE: 4|UNTERE GRENZE:16 |"
550 CHAR ,22, 9,"├───────────────┴─────────────────┤"
560 CHAR ,22,10,"|DA DER 80-ZEICHEN-BILDSCHIRM DOP-|"
570 CHAR ,22,11,"|PELT SOVIELE ZEICHEN AUFNEHMEN   |"
580 CHAR ,22,12,"|KANN WIE DER 40'ER, SIND AUCH    |"
590 CHAR ,22,13,"|SOLCH GROSSE WINDOWS, DIE SONST  |"
600 CHAR ,22,14,"|ZUVIEL PLATZ AM BILDSCHIRM BEAN- |"
610 CHAR ,22,15,"|SPRUCHEN WUERDEN, MOEGLICH.      |"
620 CHAR ,22,16,"└─────────────────────────────────┘"
630 :
640 PRINTCHR$(7);:GETKEY A$
650 :
660 REM NUN WIRD DER TEXT "UNTER DEM WINDOW" WIEDER ANGEZEIGT:
670 :
680 I=0
```

```
690 FOR Z= 4 TO 16
700 FOR S= 22 TO 56
710 AD=Z*80+S
720 SYS DEC("CDCC"),AD/256,18
730 SYS DEC("CDCC"),AD AND 255,19
740 SYS DEC("CDCC"),T(I),31
750 SYS DEC("CDCC"),1,30
760 I=I+1
770 NEXT S,Z
780 :
790 PRINTCHR$(7);:GETKEY A$
800 :
810 GOTO 500
```

Das Programm selbst ist ähnlich wie »Windowprogramm 4« aufgebaut; etwas kompliziert sind jedoch die Programmteile zum Retten und Zurückschreiben des Textes unter dem Window in den Zeilen 400-480 und 680-770. Dies liegt daran, daß der VDC (80-Zeichen-Chip) nur schwer programmierbar ist. Da die Routinen an andere Windows leicht angepaßt werden, wenn die Werte in den FOR-NEXT-Schleifen geändert werden, was kein Problem ist, soll hier auf das komplizierte Gebiet der VDC-Programmierung nicht weiter eingegangen werden. In 3.8 wird dies noch einmal aufgegriffen und ausführlich behandelt. Lassen Sie sich also von den vielen SYS- und RREG-Anweisungen in unserem letzten Beispielprogramm nicht stören.
Nun haben wir uns alle Grundlagen der Window-Programmierung angeeignet. Es soll noch einmal zusammengefaßt werden, worauf zu achten ist:

1. An der Stelle, an der das Window stehen soll, darf sich vorher kein anderer Text befinden, da dieser sonst durch das Window überschrieben wird. Eine Alternative ist das Retten des Textes an der Window-Position, das allerdings sehr langsam ist (insbesondere im 80-Zeichen-Modus).
2. Windows muß man optisch abgrenzen, damit sie vom »normalen« Bildschirm unterschieden werden können. In der Regel nimmt man Grafikzeichen, welche das Window einrahmen, oder den Reversdruck, der durch Anhängen von »,1« an den Ausgabestring beim CHAR-Befehl erzeugt werden kann.
3. Die Position des Windows muß festgelegt sein. Vor dem Ausdrucken eines Windows muß diese Window-Position als Cursorposition eingestellt werden. Hierbei ist der CHAR-Befehl eine große Hilfe.
   In der Regel wird man den Wert für »Spalte« innerhalb des Druckvorgangs zu einem bestimmten Window nicht ändern, da der linke Rand eingehalten werden muß; der Wert für »Zeile« ist jedoch dauernd um 1 hochzuzählen, wie man an unseren Beispielprogrammen verfolgen kann (beachten Sie den letzten Parameter vor dem Ausgabestring).
4. Der Einfachheit halber sollte man nur Windows in Rechteckform programmieren, weil man in diesem Fall nur zwei Begrenzungen des Windows kennen muß: Länge und Breite.
   Da das C128-Betriebssystem das Arbeiten mit Windows, wie wir es benötigen, nur unzureichend unterstützt, muß man auf die Einhaltung der Grenzen selbst achten. Bei Rechteck-Windows der gebräuchlichen Form muß man folgende Werte festlegen:

   - Grenze links (Spalte)
   - Grenze rechts (Spalte)

- Grenze oben (Zeile)
- Grenze unten (Zeile)

Man darf also, weil ein Window kleiner als der gesamte Bildschirm ist, nur eine begrenzte Textbreite verwenden; außerdem stehen nicht beliebig viele Zeilen zur Verfügung. Dies ist ganz einfach eine Frage der Planung beim Entwurf der CHAR-Zeilen. Die rechte Grenze einzuhalten ist am schwierigsten, denn hier kommt es nicht nur auf die Spalten- und Zeilenwerte bei CHAR an. Wenn man aber eine einheitliche Länge des Ausgabestrings einhält, dürfte es keine Probleme geben.

5. Mehrere Windows sind auch möglich, allerdings sollten sich die Windows nicht unbedingt überlagern, weil man sonst zu leicht den Überblick verliert (vor allem, wenn der unter einem Window stehende Text gerettet wird). Es kann nur abschließend wiederholt werden: Windows sind keine große Programmierkunst, sondern beruhen nur auf ausreichend gewissenhaftem Vorgehen des Programmierers.

3.7 Zusätzliche Befehle und Möglichkeiten

In der Überschrift dieses dritten Kapitels heißt es: »Von Basic 2.0 zu Basic 7.0 - und noch weiter«. Nachdem wir jetzt die neuen Befehle des Basic 7.0 kennengelernt haben, soll das Versprechen »und noch weiter« eingehalten werden. Wir werden den Basic-7.0-Befehlsvorrat einerseits mit Maschinenprogrammen erweitern (in Basic ist dies nicht möglich), andererseits aber weitere Möglichkeiten kennenlernen, die zusätzlichen Befehlen gleichkommen (Anwendungen des Tastaturpuffers). Folgende Befehlserweiterungen wollen wir erreichen:

Hochauflösende Grafik auf dem 80-Zeichen-Bildschirm

Dabei werden wir uns weiterhin auf die Befehle des Basic 7.0 zur Grafikprogrammierung stützen; diese erweitern wir nur um den Modus »GRAPHIC 6« (hochauflösende Grafik auf dem 80-Zeichen-Bildschirm). Multicolorgrafiken und Sprites können beim besten Willen auf dem 80er-Bildschirm nicht erstellt werden, da der VDC (80-Zeichen-Chip) dazu nicht in der Lage ist. Grafikfähig ist er aber, wie wir gleich sehen werden, und zwar mit einer doppelt so hohen Auflösung in der X-Richtung: wie der VIC (640*200 Punkte statt 320*200).

OLD

Dieser Befehl, der in manchen C64-Basic-Erweiterungen vorhanden ist, regeneriert ein durch NEW oder Reset gelöschtes Basic-Programm. Wir werden uns aber dazu kein Maschinenprogramm basteln, sondern eine viel einfachere - aber genauso wertvolle - Lösung entwickeln.

FIND

Um in einem Programm einen Text (Befehl, Variable, Text in String, Zahl, ...) zu suchen, kann dieser Befehl, den wir mit Hilfe einer Maschinenroutine simulieren, verwendet werden.

MERGE

Das Verkoppeln zweier (oder mehrerer) Programme wird als »MERGE«-Vorgang bezeichnet, aber von Basic 7.0 nicht unterstützt. Wir entwickeln zwei Lösungen: mit und ohne Maschinenroutine.

3.7.1 Hochauflösende Grafik auf dem 80-Zeichen-Bildschirm

Schon oft haben wir die Möglichkeit des VDC, hochauflösende Grafiken mit einer Auflösung von 640*200 Punkten darzustellen, angesprochen. Jetzt wird ein Maschinenprogramm vorgestellt, welches die Basic-7.0-Befehle um die Möglichkeit erweitert, in einem »GRAPHIC 6«-Modus zu arbeiten.
Gehen Sie jetzt bitte in den 80-Zeichen-Modus, legen Sie die Programmdiskette ein und erteilen Sie folgendes Kommando:

```
RUN"GRAFIK-80.DEMO 1"
```

Dann haben Sie schon einen ersten Eindruck und können das, was im folgenden besprochen wird, sicher besser verstehen.

Das Installationsprogramm und der Maschinencode

Das Programm »GRAFIK-80«, das aus Programmierersicht uninteressant und deshalb auch nicht als Listing abgedruckt ist, erzeugt auf Diskette ein File namens »GRAFIK-80.M« (das ».M« steht für »Maschinencode«). Dies ist das auf der Programmdiskette bereits enthaltene endgültige Programm, das über folgende Befehle geladen und initialisiert wird:

```
BLOAD"GRAFIK-80.M",ON B0:BANK15:SYSDEC("1303")
```

Nach diesem Kommando kann GRAPHIC außer 0-5 auch folgenden Modus anwählen:

GRAPHIC 6	Hochauflösende Grafik auf 80-Zeichen-Bildschirm
GRAPHIC 6,1	wie »GRAPHIC 6«, aber mit Löschen des Bildschirms

»GRAPHIC 6,1« ist als Ersatz für den nicht implementierten Befehl »SCNCLR (6)« zu betrachten. Nach »GRAPHIC 6« kann aber SCNCLR (ohne Parameter!) weiterhin benutzt werden.
Folgende Befehle können sich jetzt auch auf den 80-Zeichen-Bildschirm beziehen:

GRAPHIC, BOX, CIRCLE, DRAW, PAINT, LOCATE, SCALE, SCNCLR ohne Parameter

Gleiches gilt für folgende Funktionen:

RCLR, RDOT, RGR

Nicht verfügbar sind allerdings die SHAPE-Befehle, WIDTH und alle Sprite-Anweisungen. Das Programm »GRAFIK-80.M« belegt den Speicherbereich von 4864 bis 7167, der für solche

Zwecke vorgesehen ist. Probleme ergeben sich nur, wenn ein anderes Maschinenprogramm den gleichen Bereich belegt.

Flimmerproblem beseitigen

Nur mit dem Laden und Initialisieren ist es jedoch nicht getan, da zwei leicht unterschiedliche Versionen des 80-Zeichen-Chips VDC existieren. Das Programm muß auf die entsprechende Version vorbereitet sein. Hierfür gibt es zwei Möglichkeiten:

Installation für VDC-Version vornehmen

Das Installationsprogramm erstellt auf Diskette ein Maschinencode-File »GRAFIK-80.M«, welches auf Ihrem C128 unter Garantie ohne Flimmern läuft, da vor dem Abspeichern des Maschinenprogramms auf Diskette die nötigen Anpassungen vorgenommen werden.
Allerdings kann dadurch nicht verhindert werden, daß auf einem anderen C128 dieses »Kräuseln« auftritt.

Maschinencode nach Ladevorgang anpassen

Die bessere Lösung ist, das Programm nach dem Einlesen über BLOAD entsprechend an die C128-Version, auf der es gerade läuft, vorzubereiten und erst dann zu starten.
Dazu müssen nach dem Ladevorgang folgende Befehle ausgeführt werden:

```
BANK 15:SYSDEC("CDDA"),,25:RREG TS:
IF TS=71 THEN POKE 6752,71:POKE 6789,135:
ELSE POKE 6752,64:POKE 6789,128
```

In unserem Demoprogramm »GRAFIK-80.DEMO 1« steht diese Befehlsfolge in Zeile 210.
Allerdings funktioniert dies nicht, wenn bereits die hochauflösende 80-Zeichen-Grafik mit GRAPHIC 6 eingeschaltet wurde.
Sehen wir uns deshalb als Musterbeispiel das Demoprogramm an:
Beschreibung: Demoprogramm 1 für »Grafik-80«
Filename: »grafik-80.demo 1«

```
100 REM ***********************
110 REM *                     *
120 REM *  GRAFIK-80-DEMO 1   *
130 REM *                     *
140 REM ***********************
150 :
160 REM DURCH BELIEBIGEN TASTENDRUCK KANN DIE DEMO ABGEBROCHEN WERDEN.
170 :
180 IF RWINDOW(2)<>80 THEN PRINT "80-ZEICHEN-MODUS !"
190 FAST:BANK 15
200 BLOAD"GRAFIK-80.M":REM GRAFIK-80 LADEN
210 BANK15:SYSDEC("CDDA"),,25:RREG TS:IF TS=71 THEN POKE 6752,71:POKE 6789,135:E
LSE POKE 6752,64:POKE 6789,128:REM GRAFIK-80 ANPASSEN
220 SYS DEC("1303"):REM GRAFIK-80 INITIALISIEREN
230 :
240 GRAPHIC 6,1:REM HOCHAUFLOESENDE VDC-GRAFIK EINSCHALTEN
250 CIRCLE,320,100,250,99,70,260
```

```
260 BOX, 145,100, 115,110, 160, 1
270 BOX, 145,100, 115,110,  70, 1
280 CIRCLE ,110,45,40,15,1
290 CIRCLE,500,40,37,17,,,,45
300 CIRCLE,350,100, 160,60, 145,220
310 DRAW 1, 300,50 TO 300,110 TO 400,120 TO 300,50
320 CIRCLE0,320,100,250,99,70,260
330 CIRCLE,320,100,250,99,70,260
340 PAINT ,320,90
350 CIRCLE, 130,85, 200,80, 330,10
360 CIRCLE, 509,85, 200,80, 330,10
370 CIRCLE, 110,0, 100,55, 158,202
380 CIRCLE, 490,0,  97,50, 155,190
390 GET Q$: IF Q$<>"" THEN 450:REM BEI TASTENDRUCK PROGRAMM BEENDEN
400 CIRCLE1, 340,196, 160,63, 325,40
410 FOR I=1 TO 500: NEXT:REM KLEINE VERZOEGERUNGSSCHLEIFE
420 CIRCLE0, 340,196, 160,63, 325,40
430 FOR I=1 TO 500: NEXT:REM KLEINE VERZOEGERUNGSSCHLEIFE
440 GOTO390
450 GRAPHIC 5:REM HOCHAUFLOESUNG AUSSCHALTEN
460 END
```

Die Zeilen 180-220 sollten, unter Umständen mit anderen Zeilennummern versehen (aber in gleicher Reihenfolge), am Anfang jedes Programms stehen, das hochauflösende Grafik auf dem 80-Zeichen-Bildschirm bewirkt.
Ein Beispiel für die Grafikbefehle im 80-Zeichen-Modus sind die Zeilen 240-460. In Zeile 390 wird geprüft, ob eine Taste gedrückt wurde; ist dies der Fall, so wird das Programm beendet. Dabei muß mit GRAPHIC 5 die Hochauflösung abgeschaltet werden, wobei automatisch der Textbildschirm gelöscht wird, da dieser durch die hochauflösende Grafik verlorengeht - im Gegensatz zum 40-Zeichen-Bildschirm.

<RUN/STOP>+<RESTORE> wirkungslos?

Um aus der hochauflösenden VDC-Grafik in den 80-Zeichen-Textmodus zu gelangen, reicht es nicht aus, <RUN/STOP>+<RESTORE> zu drücken. Zusätzlich muß man <CAPS LOCK> verriegeln; sobald der Textbildschirm erscheint, kann <CAPS LOCK> wieder entriegelt werden, wenn man den deutschen Zeichensatz nicht benötigt.
Probieren Sie dies am besten an »GRAFIK-80.DEMO 1« aus.

Das zweite Demonstrationsprogramm

Am zweiten Demonstrationsprogramm kann auch gleich gezeigt werden, wie man die Farben der hochauflösenden 80-Zeichen-Grafik bestimmt.

Beschreibung: Demoprogramm 2 für hochauflösende VDC-Grafik
Filename: »grafik-80.demo 2«

```
100 REM **********************
110 REM *                    *
120 REM *  GRAFIK-80-DEMO 2  *
130 REM *                    *
140 REM **********************
150 :
160 PRINT CHR$(142) CHR$(11):REM GROSS/GRAFIKZEICHENSATZ EINSCHALTEN UND SHIFT C
= ABSCHALTEN
170 :
180 IF RWINDOW(2) = 40 THEN PRINT CHR$(7)"AUF 80-ZEICHEN-BILDSCHIRM UMSCHALTEN U
ND TASTE DRUECKEN !":GETKEY A$:REM SONDERBEHANDLUNG FUER 40-ZEICHEN-BILDSCHIRM
190 :
200 BANK15:SYS DEC("CDDA"),,25:RREG TS:REM WERT VON REGISTER 25 IN VARIABLE TS H
OLEN (TS = TESTVARIABLE FUER VERSION DES VDC-CHIPS, UM PROGRAMM ANZUPASSEN)
210 :
220 FAST:GRAPHIC 5,1:COLOR 6,7:COLOR 5,8:REM DOPPELTE GESCHWINDIGKEIT, GELBE ZEI
CHEN, BLAUER HINTERGRUND
230 :
240 CHAR,32,3,CHR$(2)+CHR$(7)+"DEMO 2 (ZU GRAFIK-80)"
250 CHAR,1,5,CHR$(13)
260 :
270 PRINT "DIE ERWEITERUNG 'GRAFIK-80' MUSS GELADEN UND INITIALISIERT WERDEN, DA
MIT DIE BE-"CHR$(13)
280 PRINT "FEHLE ZUR HOCHAUFLOESENDEN GRAFIK  AUF DEM 80-ZEICHEN-BILDSCHIRM  ZUR
 VERFUEGUNG"CHR$(13)
290 PRINT "STEHEN. LEGEN SIE DAZU EINE DISKETTE EIN, AUF DER BEREITS  DAS MASCHI
NENPROGRAMM"CHR$(13)
300 PRINT "'GRAFIK-80.M'  ABGESPEICHERT  IST, DAS  VOM  GENERIERUNGSPROGRAMM  ER
ZEUGT WIRD."CHR$(13)
310 PRINT CHR$(2)"DRUECKEN SIE DANN DIE RETURN- ODER ENTER-TASTE"
320 :
330 DO:GETKEY A$:LOOP UNTIL A$=CHR$(13):PRINT CHR$(7)
340 :
350 BLOAD "GRAFIK-80.M",ON B15:SYS DEC("1300") :REM "GRAFIK-80.M" (MASCHINENCODE
 VON "GRAFIK-80") LADEN UND STARTEN
360 REM NUN WIRD NOCH SICHERHEITSHALBER DAS PROGRAMM AN DIE JEWEILIGE VERSION
        DES VDC-CHIPS ANGEPASST:
370 IF TS=71 THEN POKE 6752,71:POKE 6789,135:ELSE POKE 6752,64:POKE 6789,128:REM
 DIE TESTVARIABLE TS WURDE IN ZEILE 200 ERRECHNET
380 SYS DEC("1303"):REM GRAFIK-80 INITIALISIEREN
390 :
400 GRAPHIC 6,1:REM HOCHAUFLOESENDE VDC-GRAFIK EINSCHALTEN
410 SYS DEC("CDCC"),15*16+0,26:REM WEISSE PUNKTE AUF SCHWARZEM HINTERGRUND
420 :
430 FOR F=1 TO 700:REM 700 ZUFAELLIG POSITIONIERTE PUNKTE ZEICHNEN
440 :DRAW,RND(1)*640,RND(1)*200
450 NEXT
460 :
470 PRINT CHR$(7):REM KLINGELZEICHEN
480 :
490 FOR F=0 TO 320
500 :Y=INT(RND(1)*200):REM POSITION DER 'LUECKE' IN DER LINIE ERRECHNEN
510 :DRAW,F,0 TO F,200:DRAW 0,F,Y:DRAW 1,640-F,0TO640-F,200:DRAW0,640-F,Y:REM LI
NKS UND RECHTS JE EINE LINIE ZEICHNEN UND DABEI EINE 'LUECKE' LASSEN
520 NEXT
530 :
540 SLEEP 1:PRINT CHR$(7):REM KURZE PAUSE, DANN KLINGELZEICHEN
550 :
560 FOR F=0 TO 320:REM LANGSAMER ABBAU DER GRAFIK DURCH NACH INNEN GEHENDES LOES
```

```
CHEN VON RECHTECK-UMRISSEN MITTELS BOX
570 BOX 0,F,200-F/3.2,640-F,F/3.2:REM 3.2 = 640 (PUNKTE IN X-RICHTUNG) / 200 (PU
NKTE IN Y-RICHTUNG)
580 NEXT
590 :
600 COLOR 6,7:COLOR 5,8:GRAPHIC 5:REM HOCHAUFLOESUNG AUSSCHALTEN
610 PRINT CHR$(7)"DER LANGSAME ABBAU DER GRAFIK WIRD DURCH NUR 3 BASIC-BEFEHLE B
EWIRKT :"CHR$(13)
620 PRINT "FOR I=0 TO 320"
630 PRINT TAB(4)"BOX 0,I,200-I/3.2,640-I,I/3.2"
640 PRINT "NEXT I"
650 SLEEP 20:PRINT CHR$(7)
660 GRAPHIC6,1:SYS DEC("CDCC"),13*16+2,26
670 FOR F=0 TO 150 STEP 2
680 :BOX,F*3.5,F,F+20,F+10
690 NEXT F
700 BOX,F*3.5,F,F+20,F+10,,1
710 :
720 PRINT CHR$(7):SLEEP 5
730 :
740 GRAPHIC 5:COLOR 6,7:COLOR 5,8
750 PRINT "DIE LETZTE GRAFIK WURDE MIT FOLGENDEN BEFEHLEN ERZEUGT :"CHR$(13)
760 PRINT "FOR I=0 TO 150 STEP 2"
770 PRINT TAB(4)"BOX 1,I*3.5,I,I+20,I+10"
780 PRINT "NEXT I"
790 PRINT "BOX 1,I*3.5,I,I+20,I+10,,1"
800 PRINT CHR$(13)"NAECHSTES BILD: SINUS- UND KOSINUS-KURVE"
810 SLEEP 20
820 :
830 GRAPHIC 6,1:SYS DEC("CDCC"),7*16+2,26
840 PRINT CHR$(7)
850 FOR F=0 TO 640 STEP .55
860 :DRAW,F,SIN(F/50)*90+100
870 NEXT F
880 :
890 SLEEP 1:PRINTCHR$(7)
900 :
910 FOR F=0 TO 640 STEP .55
920 :DRAW,F,COS(F/50)*90+100
930 NEXT F
940 :
950 SLEEP 5
960 :
970 GRAPHIC 5:COLOR 5,8
980 PRINT "NUN WERDEN MEHRERE FUNKTIONSKURVEN SCHEINBAR GLEICHZEITIG GEZEICHNET.
"
990 PRINT CHR$(7):SLEEP 3
1000 :
1010 GRAPHIC 6,1:SYS DEC("CDCC"),13*16+0,26
1020 :
1030 FOR F=0 TO 640 STEP .55
1040 :DRAW,F,SIN(F/200)*90+100
1050 :DRAW,F,COS(F/80)*90+100
1060 :DRAW,F,SIN(F/80)*90+100
1070 :DRAW,F,COS(F/200)*90+100
1080 :DRAW,F,SQR(F/200)*200
1090 :DRAW,F,SIN(F/60)*30+100
1100 :DRAW,F,(-SIN(F/80))*90+100
1110 :DRAW,F,(-COS(F/80))*90+100
1120 :DRAW,F,SIN(F/200)*5+100
1130 NEXT F
1140 :
1150 PRINT CHR$(7)
```

```
1160 :
1170 FOR F=1 TO 10
1180 PAINT,RND(TI)*640,RND(1)*200
1190 NEXT
1200 :
1210 PRINT CHR$(7):SLEEP 5:GRAPHIC 5
1220 PRINT "ES FOLGEN EINIGE GRAFIKEN, DIE MIT WENIGEN BASIC-ZEILEN ERZEUGT WERD
EN."
1230 SLEEP 5
1240 :
1250 GRAPHIC 6,1:SYS DEC("CDCC"),7*16+2,26
1260 :
1270 CIRCLE,320,100,130,10
1280 CIRCLE,320,100,130,50
1290 CIRCLE,120,40,40,18,,,,45
1300 CIRCLE,520,40,40,,,,,90
1310 CIRCLE,120,140,40,18,,,,120
1320 CIRCLE,520,140,40,18,,,,60
1330 :
1340 PRINT CHR$(7):SLEEP 10:GRAPHIC 5
1350 COLOR 5,8
1360 PRINT "FOLGENDE BEFEHLE HABEN DIE LETZTE GRAFIK GEZEICHNET :"
1370 PRINT CHR$(13)
1380 PRINT "CIRCLE 1,320,100,130,10"
1390 PRINT "CIRCLE 1,320,100,130,50"
1400 PRINT "CIRCLE 1,120,40,40,18,,,,45"
1410 PRINT "CIRCLE 1,520,40,40,,,,,90"
1420 PRINT "CIRCLE 1,120,140,40,18,,,,120"
1430 PRINT "CIRCLE 1,520,140,40,18,,,,60"
1440 PRINTCHR$(13)
1450 PRINT"DER CIRCLE-BEFEHL IST ALSO IN SEHR VIELFAELTIGER WEISE VERWENDBAR."
1460 SLEEP 10
1470 :
1480 GRAPHIC 6,1:SYS DEC("CDCC"),7*16+2,26
1490 :
1500 FOR F=1 TO 640 STEP 8
1510 :DRAW,F,50 TO 140,F/3.2
1520 NEXT
1530 :
1540 PRINT CHR$(7):SLEEP 10
1550 GRAPHIC 5
1560 PRINT "FOLGENDE BEFEHLE ZEICHNETEN DIE LETZTE GRAFIK :"
1570 PRINT CHR$(13)
1580 PRINT "FOR I=1 TO 640 STEP 8"
1590 PRINT TAB(4)"DRAW 1,I,50 TO 140,I/3.2"
1600 PRINT "NEXT"
1610 PRINTCHR$(7):SLEEP 10
1620 :
1630 GRAPHIC 6,1:SYS DEC("CDCC"),7*16+2
1640 :
1650 FOR F=1 TO 639 STEP 4
1660 :DRAW,F,40 TO 100*SIN(F/30)+300,F/3.2
1670 NEXT
1680 :
1690 PRINTCHR$(7):SLEEP 10
1700 GRAPHIC 5
1710 PRINT "FOLGENDE BEFEHLE ZEICHNETEN DIE LETZTE GRAFIK :"
1720 PRINT CHR$(13)
1730 PRINT "FOR I=1 TO 639 STEP 4"
1740 PRINT TAB(4)"DRAW 1,I,40 TO 100*SIN(I/30)+300,I/3.2"
1750 PRINT "NEXT"
1760 :
1770 SLEEP 10:GRAPHIC 6,1:SYS DEC("CDCC"),7*16+2,26
```

```
1780 :
1790 FOR F=1 TO 319 STEP 3
1800 :DRAW,F*2,40*COS(F/30)+100 TO 100*SIN(F/30)+300,F/1.6
1810 NEXT
1820 :
1830 PRINT CHR$(7):SLEEP 5:GRAPHIC 5
1840 PRINT "FOLGENDE BEFEHLE ZEICHNETEN DIE LETZTE GRAFIK :"
1850 PRINT CHR$(13)
1860 PRINT "FOR I=1 TO 319 STEP 3"
1870 PRINT TAB(4)"DRAW 1,I*2,40*COS(I/30)+100 TO 100*SIN(I/30)+300,I/1.6"
1880 PRINT "NEXT"
1890 :
1900 PRINT CHR$(13):REM 2 LEERZEILEN
1910 PRINT "**** ENDE DES DEMOPROGRAMMS ****"
1920 END
```

Während man die Zeichenfarbe für hochauflösende Grafik am 40er-Bildschirm mit »COLOR 1,...« bestimmt, geht dies im 80-Zeichen-Modus nur über einen SYS-Befehl. Zudem darf die Zeichenfarbe nicht für jeden 8*8-Pixel-Block neu definiert werden.
Als Beispiel dient Zeile 410. Diese stellt die Punktfarbe auf »weiß« und den Hintergrund auf »schwarz«. Der SYS-Befehl lautet allgemein formuliert so:

```
BANK15:SYSDEC("CDCC"),V*16+H,26
```

Wird »BANK15« einmal am Programmbeginn eingestellt und sonst kein BANK-Befehl verwendet, so kann »BANK15« später weggelassen werden.
Der Parameter »V« ist der Farbcode der Vordergrund-, »H« der Hintergrundfarbe. Allerdings ist dieser Farbcode nicht mit den bei COLOR üblichen Farbcodes identisch, sondern es gilt dafür Tabelle 3.15 (in Klammern der COLOR-Code):

Tabelle 3.15: *Farbcodes des VDC*

0	Schwarz	(1)
1	Mittelgrau	(13)
2	Blau	(7)
3	Hellblau	(15)
4	Grün	(6)
5	Hellgrün	(14)
6	Dunkelgrau	(12)
7	Türkis	(4)
8	Rot	(3)
9	Hellrot	(11)
10	Orange	(9)
11	Lila	(5)
12	Braun	(10)
13	Gelb	(8)
14	Hellgrau	(16)
15	Weiß	(2)

Die Farbe Orange ist allerdings nicht mit der gleichnamigen im 40-Zeichen-Modus zu vergleichen, sondern eher dunkellila.
Experimentieren Sie ohne weiteres mit »Grafik-80.Demo 2«, denn so können Sie am leichtesten Erfahrung in der Programmierung von hochauflösender VDC-Grafik sammeln.

3.7.2 OLD - Das Gegenstück zu NEW

Die einfachste Lösung, ein durch NEW oder einen Reset gelöschtes Basic-Programm wiederzubeleben, ist die folgende Zeile, die im Direktmodus einzugeben ist:

```
POKE PEEK(45)+256*PEEK(46),1:DELETE 1
```

Danach ist das Programm voll funktionsfähig (LIST, SAVE, RUN etc.). Diesen Einzeiler sollte man aber nicht eingeben, wenn das Programm noch im Speicher steht (mit LIST kann man dies leicht feststellen).
Trotz »DELETE 1« wird auch die Zeile 1 wiederhergestellt, womit wir schon - für Interessierte - bei der Funktionsweise wären.
Durch den POKE-Befehl wird an den Basic-Start, der mit dem PEEK-Ausdruck ermittelt wird, eine 1 geschrieben, um einigen Routinen des Computers vorzutäuschen, daß sich im Speicher ein Programm befindet. Dies allein genügt jedoch nicht, da die Zeilen noch nicht gekoppelt und daher LIST, SAVE etc. noch nicht voll funktionsfähig sind.
Erstaunlicherweise hilft hier der DELETE-Befehl weiter, der normalerweise zum Löschen von Zeilen und nicht zum Regenerieren dient.
Dieser würde ohne den vorausgegangenen POKE-Befehl erkennen, daß im Speicher kein ungelöschtes Programm steht und wirkungslos bleiben. So aber versucht er, Zeile 1 zu löschen (statt 1 kann man auch andere Zeilennummern einsetzen), die nicht vorhanden ist (in Wirklichkeit ist das Basic-Programm ja gelöscht). Nach dem versuchten Löschen wird das Programm wieder gebunden und die Basic-Zeiger auf Anfangs- und Endadresse des Programms werden ordnungsgemäß gesetzt, was in der Vorgehensweise des DELETE-Befehls begründet ist. Da RENUMBER ähnlich arbeitet, kann man »DELETE 1« auch durch »RENUMBER« ersetzen:

```
POKE PEEK(45)+256*PEEK(46),1:RENUMBER
```

So wie bei DELETE der eigentliche DELETE-Befehl (das Löschen von Zeilen) nicht ausgeführt wird, solange man das Vorhandensein eines Programms nur mittels POKE-Befehl vortäuscht, wird ebenso bei RENUMBER nicht umnumeriert, sondern nur das Programm wiederhergestellt. Gibt man allerdings den OLD-Einzeiler ein, nachdem ein Programm bereits wieder ins Leben gerufen wurde, kommt die DELETE- bzw. RENUMBER-Wirkung zum Tragen. Deshalb sollte man die vorgestellten Lösungen wirklich nur anwenden, wenn das Programm gelöscht ist.
Den Koppelvorgang kann man auch durch Aufruf der entsprechenden ROM-Routinen ersetzen:

```
POKE PEEK(45)+256*PEEK(46),1:BANK15:SYS 20303:SYS 20354
```

Dem ist der erstgenannte Vorschlag (mit DELETE bzw. RENUMBER statt SYS) gleichwertig, aber wesentlich einfacher zu merken. Andererseits sind die SYS-Anweisungen leicht als JSR-Befehle in Maschinensprache übersetzt. So sieht »OLD« in Assembler aus:

```
LDY #0         ; Offset = 0
LDA #1         ; 1 als Wert in
STA ($2D),Y ; Adresse schreiben
JSR $4F4F    ; Koppel-Routine 1
JSR $4F82    ; Koppel-Routine 2
RTS            ; Ende
```

Übrigens: Wer sich für die Funktionsweise von DELETE, RENUMBER und anderen ROM-Routinen interessiert, dem sei das Buch »C128 ROM-Listing« von Dr. Ruprecht aus der Commodore-Sachbuchreihe, vertrieben von Markt & Technik, Nummer MT 90212, wärmstens empfohlen, wo die beiden Koppel-Routinen auf Seite 84 dokumentiert werden.

3.7.3 FIND - Suchen leicht gemacht

Obwohl das Basic 7.0 des C128 viele Befehle beinhaltet, so gibt es doch einige, die man vermißt. Ein solcher ist der von vielen Basic-Erweiterungen zum C64 her bekannte Befehl FIND. Diesen bringen Sie dem C128 bei, indem Sie die Programmdiskette einlegen und folgenden Befehl eingeben: BOOT »FIND $1300«,ON B15
Floppy-1541-Besitzer schreiben bitte:
BLOAD»FIND $1300«,ON B15:BANK 15:SYSDEC(»1300«)
Dabei sollten keine anderen Basic-Erweiterungen (Grafik-80 etc.) im Speicher stehen. FIND liegt im Bereich 4864-5120.
Dann können Sie Ihr Basic-Programm nach einem Begriff durchsuchen lassen. Als Befehlserkennung dient der Klammeraffe (@). Hinter diesen schreiben Sie das Suchwort (zum Beispiel einen Befehl, einen String oder einen Variablennamen). Wenn eine Zeile gefunden wird, in der der Begriff auftaucht, so wird die komplette Zeile auf den Bildschirm gelistet (nach »OPEN 1,4:CMD 1« auf den Drucker mit der Gerätenummer 4).
Das Listen kann mit <CBM> verlangsamt und mit <NO SCROLL> angehalten werden. Vor dem Klammeraffen, der auch in Programmen vorkommen darf (ob er dort einen Sinn hat, ist eine andere Frage), muß ein Doppelpunkt stehen, wenn er am Anfang einer Bildschirmzeile befindlich ist. Beispiele:

:@DATA	listet alle DATA-Zeilen
:@GOTO 10	listet alle Zeilen, in denen GOTO 10 vorkommt, aber auch GOTO 100, GOTO 1000 etc.
:@"MESSUNG"	listet alle Zeilen, in denen der String steht.
:@CHR$(7)	listet alle Zeilen, in denen »CHR$(7)« steht.
:@45	listet alle Zeilen, in denen die Zahl 45 steht.

:@"KOMMENTAR" listet alle Zeilen, in denen der Kommentar als REM-Anmerkung oder in einem String vorkommt.

Wird das zweite Anführungszeichen weggelassen, meldet der C128 einen »?SYNTAX ERROR«.

3.7.4 MERGE - Programme koppeln

Sollen zwei Programme verbunden werden (z.B. um eine häufig benötigte Unterroutine wie eine der in 3.7 vorgestellten Routinen einzubinden), so kann dies auf zwei Arten geschehen. Entweder man arbeitet mit PEEK und POKE, oder man bedient sich einer Hilfsroutine in Maschinensprache, die die PEEK- und POKE-Operationen übernimmt.

MERGE mit PEEK und POKE

An einem Beispiel, das zwar unsinnig ist, aber eben nur der Demonstration dienen soll, kann das MERGE-Verfahren mit PEEK und POKE erklärt werden. Wir wollen die Programme »GRAFIK-80.DEMO 1« und »GRAFIK-80.DEMO 2« miteinander verbinden. Dabei stellt sich als erstes das Problem, daß beide Programme im selben Zeilenbereich liegen (100 ist erste Zeilennummer, danach geht es in 10er-Schritten weiter). Folglich muß ein RENUMBER erfolgen. Wir laden zuerst »GRAFIK-80.DEMO 1« (Programmdiskette einlegen!):

```
DLOAD "GRAFIK-80.DEMO 1"
```

Dann numerieren wir dieses Programm neu:

```
RENUMBER 0,1
```

Jetzt liegt es im Zeilenbereich 0-36, und das zweite Programm kann weiterhin ab Zeile 100 stehen, ohne daß es eine verhängnisvolle Überschneidung gibt. Wenn das erste Programm ohnehin in einem niedrigeren Zeilenbereich als das zweite liegt, so muß natürlich kein RENUMBER erfolgen.
Als nächstes lassen wir uns drei Werte ausrechnen, die wir uns merken müssen (aufschreiben oder mit Hilfe der Window-Definition am Bildschirm schützen):

```
PRINT PEEK(45),PEEK(46),PEEK(4624)+256*PEEK(4625)
```

Die Befehle sollte man abkürzen (»?« für »PRINT«, »P<SHIFT>+<E>für »PEEK«), damit diese Eingabe auch am 40-Zeichen-Bildschirm in eine einzige Bildschirmzeile paßt.
Wir erhalten im Beispiel folgende Werte:

```
1        28        8374
```

Vom dritten Wert subtrahieren wir 2 und erhalten 8372.
Dieses Subtraktionsergebnis müssen wir folgendermaßen verwerten:

```
POKE 45,8372 AND 255:POKE 46,8372/256
```

Für Insider: Low- und High-Byte werden im Zeiger auf den Anfang des Basic-Programms im Speicher abgelegt.
Dann nehmen wir wieder den ursprünglichen dritten Wert (also vor dem Abziehen von 2):

```
POKE 8374,0
```

Nun kann das anzuhängende Programm nachgeladen werden, ohne daß das erste Programm verlorengeht:

```
DLOAD "GRAFIK-80.DEMO 2"
```

Auch die beiden ersten am Anfang ermittelten Werte (1 und 28) sollen jetzt eingesetzt werden:

```
POKE 45,1:POKE 46,28
```

Dann sind die beiden Programme verbunden und stehen im Speicher; sie können abgespeichert, gelistet oder gestartet werden. Dieses Beispiel ist zwar, wie schon gesagt, nicht besonders nützlich, aber es ist ja auch nur ein Beispiel – nicht mehr und nicht weniger.
Beim C64 funktioniert diese Art von MERGE auch; die Adresse 45 ist durch 43, 46 durch 44, 4625 durch 45 und 4626 durch 46 zu ersetzen.

MERGE mit Maschinenprogramm

Etwas einfacher geht es mit Verwendung eines Maschinenprogramms, das auf der Programmdiskette steht und durch folgenden Befehl geladen wird:

```
BLOAD"MERGE.2816/2848",ON B15
```

Nach dem Laden des ersten Programms geben Sie dann

```
SYS 2816
```

ein, laden das zweite Programm und verbinden beide über

```
SYS 2848
```

Danach können Sie abspeichern, starten oder listen.
Im Prinzip werden die POKE-Befehle durch den Einsatz dieser kurzen Routine, die im Kassettenpuffer (2816-2859) steht (bei RS232/Kassetten-Betrieb: Programmverlust!) und aus zwei Teilen besteht (Teil 1 wird mit »SYS 2816«, Teil 2 mit »SYS 2848« gestartet), eingespart. Die Routine ist allerdings ebensowenig wie die PEEK- und POKE-Befehle gegen Fehleingaben abgesichert. Seien Sie also gewissenhaft.
Das Maschinenprogramm kann mit Basic-Erweiterungen (Grafik-80, FIND etc.) problemlos betrieben werden, da es nur über SYS aufgerufen, nicht aber ins Basic 7.0 eingebunden wird.
Hier noch der Quelltext für diejenigen, die Maschinensprachekenntnisse besitzen und sich für die Funktionsweise interessieren:

Beschreibung: Quelltext zur MERGE-Routine
Filename: »merge.src«

```
READY.
100  -.BASE $B00                          ; IM RS-232-PUFFER ABLEGEN
110  -;
120  -; *** HILFSROUTINEN FUER 'MERGE' ***
130  -; *** ========================== ***
140  -;
150  -.DEFINE BASICANFANG          = $2D    ; ZEIGER DES
160  -.DEFINE BASICENDE            = $1210  ; BASIC-INTERPRETERS
170  -;
180  -                    LDA BASICANFANG; ZWEITE ROUTINE
190  -                    STA LOWBYTE+1  ; GEMAESS  ANFANG
200  -                    LDA BASICANFANG+1; DES BASICPROGRAMMS
210  -                    STA HIGHBYTE+1 ; MODIFIZIEREN
220  -;
230  -                    LDA BASICENDE  ; BASICENDE-2
240  -                    SEC            ; WIRD NUN ALS
250  -                    SBC #<(2)      ; BASICANFANG
260  -                    STA BASICANFANG; IN DIE
270  -                    LDA BASICENDE+1; DAFUER
280  -                    SBC #>(2)      ; VORGESEHENEN
290  -                    STA BASICANFANG+1; ZEIGER GESCHRIEBEN
300  -;
310  -                    LDY #2         ; 2 ALS OFFSET
320  -                    LDA #0         ; 0-BYTE
330  -                    STA (BASICANFANG),Y; SCHREIBEN
340  -                    RTS            ; ENDE DER ERSTEN ROUTINE
350  -;
360  -LOWBYTE             LDA #0         ; 0 = DUMMY (WIRD MODIFIZIERT)
370  -                    STA BASICANFANG; BASICANFANG VOR AUFRUF
380  -HIGHBYTE            LDA #0         ; DER 1. MERGE-ROUTINE
390  -                    STA BASICANFANG+1;WIEDERHERSTELLEN
400  -                    RTS            ; ENDE DER ZWEITEN ROUTINE
410  -;
```

Dieser Quelltext wurde übrigens, wie fast alle anderen Quelltext in diesem Buch auch, mit dem Assembler TOP-ASS (Markt & Technik) erstellt, da dieser bislang unübertroffen ist.
Über DLOAD können Sie das File zwar einladen, aber nur mit TOP-ASS listen, editieren und neu-assemblieren.

3.7.5 Anwendungen des Tastaturpuffers

Hört sich an wie ein Widerspruch: Programmierter Direktmodus. Damit ist gemeint, daß der Computer innerhalb eines Programms kurzfristig in den Eingabezustand (Direktmodus) versetzt wird, dort Eingaben tätigt, die vorprogrammiert wurden, und dann wieder ins Programm zurückspringt. Dies ist aufgrund der sogenannten »dynamischen Tastaturabfrage« möglich. Was es damit auf sich hat, können wir testen, wenn Sie folgende Eingabe machen und nach Betätigen von <RETURN> unmittelbar die Taste <F1> drücken:

```
SLEEP 5
```

Nachdem SLEEP fünf Sekunden gewartet hat, erscheint der Text GRAPHIC, obwohl dieser lange vor dem Wiedererscheinen des Cursors mittels <F1> eingegeben wurde. Man möchte meinen, daß SLEEP den Computer wirklich anhält und in dieser Zeit auch keine Tastaturabfrage erfolgt; dem ist aber nicht so. Die Tastatur wird nämlich ununterbrochen behandelt (Ausnahme: Diskettenoperationen DLOAD, DSAVE usw.). Sobald dann der Cursor wieder sichtbar ist, werden die vorher getätigten Tastendrücke - im Beispiel <F1> - verarbeitet. Diese unabhängige und immer stattfindende Tastaturabfrage wird als »dynamisch« bezeichnet.
Wenn Sie schnell tippen, können Sie in den 5 SLEEP-Sekunden auch mehr als nur einen Tastendruck eingeben, wobei Sie nicht auf die Funktionstasten beschränkt sind. Allerdings sind maximal 10 Tastendrücke möglich, unabhängig von der SLEEP-Zeit.
Wie funktioniert nun die »dynamische Tastaturabfrage«?
Ganz einfach: eine Routine, die jede 1/60 Sekunde unabhängig vom Hauptprogramm abläuft, fragt die Tastatur ab und speichert jeden Tastendruck im Bereich 842-851. Da dieser Speicher, »Tastaturpuffer« genannt, 10 Bytes groß ist, kann er auch nur 10 Zeichen speichern, weshalb nur 10 Tastendrücke während der SLEEP-Warteschleife möglich sind.
Wie viele Zeichen im Tastaturpuffer stehen, sagt Adresse 208 aus. Deshalb können mit »POKE 208,0« alle vorher erfolgten Tastendrücke gelöscht werden, was manchmal ganz nützlich ist. Aber es kann auch durch POKEn in 208 vorgetäuscht werden, daß eine bestimmte Anzahl von Tastendrücken erfolgt ist. Da der Tastaturpuffer (Adressen 842-851) über POKE ansprechbar ist, kann man auf diese Weise bestimmte Tastendrücke simulieren. Diese kommen aber nur zur Geltung, wenn sich der Computer im Eingabemodus (INPUT/GETKEY oder Direktmodus) befindet, da nur in dieser Situation der Tastaturpuffer verarbeitet wird. Dieser wird dann Zeichen für Zeichen entleert, das nächste Zeichen steht immer in 842.
Als Beispiel wollen wir die Eingabe des Buchstaben A, der den ASCII-Code 65 hat (im Tastaturpuffer stehen alle Werte im ASCII-Code), vortäuschen:

```
POKE 842,ASC("A"):POKE 208,1      oder:      POKE 842,65:POKE 208,1
```

Auf diese Befehle hin, die in Adresse 842 den Buchstaben A schreiben und die Anzahl der Zeichen im Tastaturpuffer auf 1 setzen, erscheint »READY.«, aber gleich darunter das A, wie wenn es per Tastatur eingegeben worden wäre.
Auch zwei Tastendrucke können simuliert werden:

```
POKE 842,ASC(A"):POKE 843,ASC("B"):POKE 208,2
```

Wie Sie sehen, kommt der zweite Tastendruck nach 843, der dritte nach 844, ... und der zehnte nach 851 - mehr geht nicht.
Die Adresse 208 hat die Anzahl der simulierten Tastendrucke zu enthalten.
An dieser Stelle sei gesagt, daß das Einstellen der Speicherbank überflüssig ist, da die Adressen 208/842-851 im Bereich 0-1023, der allen Banks gemeinsam ist, liegen.
Nun können wir auch das Drücken der Steuertasten simulieren, deren ASCII-Codes ja in der »Commodore-128-Codetabelle« ab Seite A-7 im C128-Handbuch zu finden sind. Hier seien die wichtigsten herausgegriffen:

13 <RETURN> zur Bestätigung einer Eingabe
18 <HOME> zum Positionieren des Cursors links oben

20	<DEL>	zum Löschen des letzten Zeichens
27	<ESC>	(muß von ESC-Kommando gefolgt sein)
147	<CLR>	zum Löschen des Bildschirms

Als Beispiel wollen wir <A><RETURN> simulieren:

```
POKE 842,ASC("A"):POKE 843,13:POKE 208,2
```

Das ergibt einen »?SYNTAX ERROR«, weil »A« kein Basic-Befehl ist.
Nachdem wir jetzt das Prinzip der Tastaturpuffer-Manipulation besprochen haben, wollen wir interessante Anwendungen kennenlernen.

GOTO X - GOTO an errechnete Zeile

Wenn man in die oberste Bildschirmzeile einen Basic-Befehl schreibt (über PRINT oder CHAR), können mit

```
POKE 842,19:POKE 843,13:POKE 208,2:END
```

die Befehle in dieser Zeile ausgeführt werden. Ein Rücksprung ins laufende Programm ist nur möglich, wenn in der Zeile auch ein GOTO-Befehl steht.
Als Beispiel soll ein GOTO auf die in der Variablen X enthaltene Zeilennummer stattfinden:

```
CHAR,0,0,"GOTO"+STR$(X):POKE 842,19:POKE 843,13:POKE 208,2:END
```

Dieser Befehl in einem Programm ist somit eine GOTO-X-Simulation, allerdings ist die Routine »LABEL 128« (siehe 3.4.2) wesentlich vielseitiger und komfortabler. Dies sollte aber auch nur ein Beispiel sein.

Zeilen löschen oder einfügen

Die Zeile 100 kann durch folgende Anweisung, die ebenso wie »GOTO X« auch in einem Programm stehen darf, gelöscht werden:

```
CHAR,0,0,"100":POKE 842,19:POKE 843,13:POKE 208,2:END
```

Wenn hinter der Zeilennummer ein Text steht, wird die Zeile 100 geändert oder neu eingefügt. Der Rücksprung ins Programm muß dadurch geschehen, daß ein GOTO-Befehl so am Bildschirm plaziert wird, daß dieser unmittelbar unter der Zeilennummer steht. Dann muß ein zweites <RETURN> simuliert werden, welches dann auch nach dem Löschen der Zeile den GOTO-Befehl ausführen läßt.
Als Beispiel mag ein Funktionsplotter dienen. Dieser läßt eine Funktion eingeben, wandelt diese in eine DEF-FN-Zeile um, diese wird ins Programm über simulierte Tastendrücke aufgenommen und schließlich durch einen GOTO-Befehl ins Programm zurückgesprungen. Beim C128 ist dies übrigens im Gegensatz zum C64 ohne Verlust der Basic-Variablen möglich.
Hier das Programm:
Beschreibung: Funktionsplotter mit hochauflösender Grafik
Filename: »funktionsplotter«

```
10 REM *** FUNKTIONSPLOTTER ***
20 POKE0,255:POKE1,0:PRINTCHR$(142)CHR$(11):XX=0:O=1:REM GRAFIK-LOESCHEN-FLAG SE
TZEN
30 GRAPHIC 0,1:PRINT"FUNKTION : ";:OPEN 1,0:INPUT#1,F$:CLOSE 1
40 SCNCLR:PRINT"Y = ";F$;CHR$(27);"Q"
50 PRINT:PRINT"RICHTIG (J/N) ? ";:POKE 2599,0:DO:GETKEY A$:LOOP UNTIL A$="J" OR
A$="N":POKE 2599,1:PRINT A$
60 IF A$="N" THEN RUN
70 REM *** FUNKTIONSDEFINITION IN ZEILE 90 SCHREIBEN ***
80 SCNCLR:FAST:PRINT "90 DEF FNA(Q)=";F$:PRINT"GOTO 90":POKE 842,19:POKE 843,13:
POKE 844,13:POKE 208,3:END
90 DEF FNA(Q)=COS(X)+COS(X↑2)
100 IF XX THEN SLOW:GOTO 170:ELSE SCNCLR:SLOW:PRINT:PRINT:INPUT"MA■STAB ";SC
110 PRINT:PRINT"KOORDINATENSYSTEM"
120 PRINT:PRINTSPC(3)"*"SPC(8)"*"SPC(9)"*"SPC(17)"1  *"SPC(8)"*  2"SPC(6)"*  3"
130 PRINT"********"SPC(4)"*"SPC(9)"********"SPC(13)"*"SPC(8)"*"SPC(9)"* "
140 PRINTSPC(3)"*"SPC(8)"*"SPC(9)"*"SPC(20)"*"SPC(8)"********* *"
150 SLOW:DO:GETKEY A$:A%=VAL(A$):LOOP WHILE A%<1 OR A%>3
160 :
170 GRAPHIC5,1:PRINT"Y = ";F$:PRINT"MA■STAB :";SC:PRINT"KOORDINATENSYSTEM :";A%:
GRAPHIC 1,O:REM IN "O" STEHT, OB GRAFIK GELOESCHT WERDEN SOLL (O=1:JA,O=0:NEIN)
180 ON A% GOTO 380,400,420:REM RICHTIGES KOORDINATENSYSTEM ZEICHNEN (1,2 ODER 3)
190 :
200 :
210 Z=0
220 DO
230 X=(-AX+Z)/SC
240 TRAP270
250 Y=SC*(FNA(X))
260 DRAW1,Z,AY-Y
270 Z=Z+1
280 LOOPWHILEZ<320
290 COLOR 4,RCLR(4)+1 AND 15
300 GETKEY A$:GRAPHIC 0,1:COLOR 4,RCLR(4)-1 AND 15
310 PRINT"WERTE SIEHE 80-ZEICHEN-BILDSCHIRM !";CHR$(13)
320 PRINT "1. GRAFIK NOCHMAL ANZEIGEN";CHR$(13):PRINT "2. NEUER MA■STAB";CHR$(13
):PRINT "3. NEUE FUNKTION IM ALTEN MA■STAB";CHR$(13):PRINT "4. PROGRAMMENDE"
330 DO:GETKEY A$:LOOP WHILE A$<"1" OR A$>"4"
340 IF A$="1" THEN GRAPHIC 1:GOTO 290
350 IF A$="2" THEN O= 1:XX= 0:GOTO 100
360 IF A$="3" THEN O= 0:XX=-1:GOTO 30
370 PRINT "NEUSTART ┘BER F6 !":END
380 AY=100:AX=160:DRAW1,0,AYTO320,AY:DRAW1,AX,0TOAX,200
390 GOTO 430
400 AY=199:AX=0:DRAW1,AX,0TOAX,AY:DRAW1,AX,AYTO320,AY
410 GOTO 430
420 AY=100:AX=0:DRAW1,AX,0TOAX,200:DRAW1,AX,AYTO320,AY
430 GOSUB 450:IFO<>0THEN GOSUB 540
440 GOTO 190
450 WE =320/SC
460 IFWE >=230THENM=50
470 IFWE >=200ANDWE <230THENM=25
480 IFWE >=120ANDWE <200THENM=20
490 IFWE >= 65ANDWE <120THENM=10
500 IFWE >= 26ANDWE < 65THENM=5
510 IFWE >= 13ANDWE < 26THENM=2
520 IFWE<13THENM=1:F=1
530 RETURN
540 YA=AY:    XA=AX:MC=0:IFM>1THENF=1
550 DO
560 FORD=1TOM
570 XA=XA+SC:YA=YA-SC:IF(XA/F)>312THENEXIT
580 NEXTD:MC=MC+1
```

```
590 IF(XA/F/8)-1>39THEN610
600 DRAW1,(XA/F),AYTO(XA/F),(AY-5):CHAR1,(XA/F/8-1),(AY/8)-1,STR$(MC*M)
610 IF(YA/F/8)<1THEN630
620 DRAW1,AX,(YA/F)TO(AX+5),(YA/F):CHAR1,(AX/8+1),(YA/F/8),STR$(MC*M)
630 LOOP
640 RETURN
```

In Zeile 80 steht der entscheidende Teil. Ab Zeile 100 wird dann die Funktion gezeichnet, wobei eine Reihe von Einstellmöglichkeiten gegeben ist. Das Programm ist für den Betrieb mit zwei Bildschirmen konzipiert (40- und 80-Zeichen-Bildschirm), da am 80er-Bildschirm Funktion und Maßstab angezeigt werden, kann aber auch nur mit einem 40er-Bildschirm betrieben werden, an dem die Eingaben getätigt und die Funktionsgraphen dargestellt werden.
Die eigentliche Zeichenroutine steht in den Zeilen 200-280. Nach dem Zeichnen der Funktion wird die Rahmenfarbe geändert und auf einen Tastendruck gewartet. Dann verschwindet die Grafik zugunsten eines Menüs, das nicht weiter beschrieben werden muß, weil es so leicht verständlich ist.
Wenn Sie die Grafik ausgeben wollen, ist vielleicht eine der in 3.8 vorgestellten Hardcopy-Routinen das Richtige für Sie.

RENUMBER bei Tastaturpuffer-Manipulationen

Ein RENUMBER-Befehl kann Tastaturpuffermanipulationen wie simulierte GOTOs usw. nicht anpassen. Deshalb sollte man möglichst bei der Entwicklung solcher Programme auf RENUMBER verzichten, so schwer es auch fallen mag. Man kann aber das Programm RENUMBERn und dann die entsprechenden Stellen von Hand anpassen.

Adressen des Tastaturpuffers beim C64

Die Anzahl der Zeichen im Tastaturpuffer enthält beim C64 die Adresse 198, der Tastaturpuffer selbst liegt im Bereich 631-640.
Da der C64 beim Einfügen einer neuen Zeile die Variablen löscht, kann es kleine Probleme geben; C64-Tastaturpuffer-Anwendungen auf den C128 zu übertragen, ist aber meist möglich.

Sprite-DATAs berechnen

Folgendes Programm, das sich auf der Programmservice-Diskette unter dem Filenamen »SPRITE-DATAS« befindet, wandelt ein Sprite, dessen Nummer angegeben werden muß, in zwei DATA-Zeilen (32000 und 32001) mit 64 Bytes um:

```
100 do:input"spritenummer";sn:a=dec("e00")+64*(sn-1):e=a+63:
    loop while sn<1 or sn>8 or sn<>int(sn)
110 print"anfangsadresse im speicher";a
120 print"endadresse";e
130 z=32000:bank15
140 scnclr:print:print:c=0
150 print z;"data";
160 c=c+1:if a>e then print chr$(20):sz=220:goto 200
```

```
170 if c=35 then printchr$(20):z=z+1:c=0:goto 150
180 s$=str$(peek(a)):if peek(a) then
    print right$(s$,len(s$)-1);",";:a=a+1:else print",";:a=a+1
190 goto 160
200 printchr$(27)"d";"goto";sz
210 printchr$(19);:forx=1to10:poke841+x,13:next:poke208,10:end
220 scnclr:print"fertig.":poke208,0:list 32000-32001
```

Da das Programm auch die Anfangs- und Endadresse im Speicher angibt, kann die READ-POKE-Schleife leicht programmiert werden:

```
FOR I=Anfangsadresse TO Endadresse:READ A:POKE I,A:NEXT I
```

Die Zeilen außer den Sprite-Daten können mit

```
DELETE-31999
```

entfernt werden.

3.8 Weitere Tips & Tricks zum Basic 7.0

Dieser Abschnitt vermittelt noch Informationen zu unterschiedlichen Themen, die allesamt recht interessant sind.

3.8.1 Programmierung des VDC

Während die VIC-Register – wenn auch im C128-Modus mit Einschränkungen – über PEEK und POKE direkt angesprochen werden können, müssen die Register des VDC anders gesteuert werden. Eine Zusammenstellung der Register finden Sie im C128-Handbuch auf den Seiten E-4 bis E-7.
Dort wird auch eine Möglichkeit zum Setzen eines Registers vorgeschlagen, die fehlerhaft ist. Folgender Befehl schreibt in ein Register mit der Nummer »x« den Wert »a«:

```
BANK 15:SYS DEC("CDCC"),a,x
```

Beispiele haben wir im bisherigen Verlauf des Buches zur Genüge kennengelernt (Setzen der Farben bei »Grafik-80« etc.).
Den Inhalt eines Registers »x« holen folgende Befehle in die Variable A, wobei beim SYS-Befehl beide Kommas erforderlich sind:

```
BANK 15:SYS DEC("CDDA"),,x:RREG A
```

Dies ist also nur etwas umständlicher als PEEK und POKE. Wesentlich mehr Kopfzerbrechen

kann das Zugreifen auf den Bildschirmspeicher bereiten, doch auch dafür sollen Fertiglösungen vorgestellt werden.
Die Adresse im Bildschirmspeicher errechnet sich durch die Formel

```
Zeile*80 + Spalte
```

Zeile liegt im Bereich 0-24, Spalte im Bereich 0-79.

Zum Vergleich: 1024+ Zeile*40 + Spalte im 40-Zeichen-Modus, wobei Spalte nur im Bereich 0-39 liegt und 1024 die Basisadresse des Bildschirmspeichers ist (der VDC-Bildschirmspeicher beginnt bei Adresse 0 im VDC-RAM und muß also nicht addiert werden, da »+0« kein Ergebnis verändert).
Diese Adresse im Bildschirmspeicher muß jetzt in der Variablen A, der zu schreibende Wert (Bildschirmcode) in W. Dann schreibt man bei eingeschalteter Bank 15:

```
SYS DEC("CDCC"),A/256,18
SYS DEC("CDCC"),A AND 255,19
SYS DEC("CDCC"),W,31
SYS DEC("CDCC"),1,30
```

Ein Beispiel finden Sie in »Windowprogramm 5« (3.6.7) in den Zeilen 660-780.
Das Auslesen eines Wertes geht ähnlich. Die Adresse muß in A stehen, der Wert kommt in die Variable W:

```
SYS DEC("CDCC"),A/256,18
SYS DEC("CDCC"),A AND 255,19
SYS DEC("CDDA"),,31:RREG W
```

Beispiel: Zeilen 380-490 in »Windowprogramm 5« (3.6.7).
Im VDC-RAM stehen folgende Bereiche:

```
    0 - 1999    Bildschirmspeicher (wie eben besprochen)
 2048 - 4047    Attribut-RAM
 8192 - 16383   Zeichensatz (Character Generator)
```

Die Adressen 2000-2047 und 4048-8191 sind ungenutzt; mit ihrer Hilfe ist es aber möglich, noch bis zu 4 zusätzliche Zeilen am 80-Zeichen-Bildschirm darzustellen. Da dies sehr speziell ist, wird es hier nicht weiter besprochen, aber in der Ausgabe 6/86 des 64'er-Magazins finden Sie auf Seite 83 ein Programm, das davon Gebrauch macht, um die Funktionstastenbelegung außerhalb des normalen Bildschirmbereichs darzustellen. Das Betriebssystem CP/M (siehe Kapitel 6) nutzt übrigens eine auf solche Weise erzeugte 26. Zeile als Statuszeile.
Das Attribut-RAM entspricht in etwa dem Farb-RAM. Der Zeichensatz ist so organisiert wie im 40-Zeichen-Modus. Da dieser im VDC-RAM steht, ist er frei verfügbar und kann dort leicht manipuliert werden. Beim Umschalten auf den DIN- oder ASCII-Zeichensatz wird dieser übrigens jeweils in die Adressen 8192-16383 des VDC-RAM kopiert, was hauptsächlich für die zeitliche Verzögerung beim Zeichensatzwechsel verantwortlich ist. Bei hochauflösender Grafik am 80-Zeichen-Bildschirm wird praktisch das ganze VDC-RAM als Bitmap benutzt.

3.8.2 Hardcopy-Routinen in Basic 7.0 und Assembler

Hardcopy vom Textbildschirm im 40-Zeichen-Modus

Wenn man einen Bildschirminhalt schwarz auf weiß haben will, kann man einen Drucker anschließen und den Bildschirminhalt mit Hilfe eines dafür geeigneten Programms ausgeben. Wie man solche Hardcopy-Programme (für Text und Grafik) schreibt, soll hier besprochen werden. Dabei kann natürlich nicht auf jeden Druckertyp eingegangen werden.
Der Algorithmus ist klar: Bildschirmcodes werden aus dem Bildschirmspeicher geholt, in den ASCII-Code des Druckers gewandelt und ausgegeben. Alle 40 Zeichen muß ein <RETURN> (ASCII-Code 13) gesendet werden, um die Zeile abzuschließen. Vor reversen Zeichen muß <RVS ON> (ASCII-Code 18) gesendet werden, ansonsten <RVS OFF> (ASCII-Code 146). Im Anführungszeichenmodus (Quote Mode) würde auch der Drucker die Steuerzeichen ignorieren, weshalb dies geprüft werden muß. Folgendes Programm gibt eine Hardcopy aus und funktioniert auf jeden Fall auf Commodore-Druckern (MPS 801, 802 und 803) und den meisten anderen, da die verwendeten Steuerzeichen, Geräte- und Sekundäradressen die gebräuchlichsten sind:
Beschreibung: Hardcopy vom 40-Zeichen-Textbildschirm
Filename: »hardcopy(text40)«

```
50000 REM ****************************
50010 REM **                        **
50020 REM ** ROUTINE FUER HARDCOPIES **
50030 REM **                        **
50040 REM **   VOM TEXT-BILDSCHIRM   **
50050 REM **                        **
50060 REM **   IM 40-ZEICHEN-MODUS   **
50070 REM **                        **
50080 REM ****************************
50090 :
50100 REM VERWENDETE VARIABLEN:
50110 REM =====================
50120 REM
50130 REM AN,AS,BC,RO,RV,SP,ZL
50140 REM ====================
50150 REM
50160 :
50170 FAST:OPEN 1,4
50180 FOR ZL = 0 TO 24
50190 RV = 0:RO = -1:AN = 0:PRINT#1,CHR$(13);
50200 FOR SP = 0 TO 39
50210 BC = PEEK (1024 + ZL*40 + SP):RV=(BC>127)AND(NOT RV):BC=BCAND127
50220 RO=(NOT(PEEK(1024+ZL*40+SP)>127))AND(NOT RO)
50230 AS = BC+64+64*(BC<64ANDBC>31)+32*(BC<96ANDBC>63)
50240 IF AN AND (NOT RV) AND PEEK (1024+ZL*40+SP) = 32 THEN 50280
50250 IF RO AND (NOT AN) THEN PRINT#1,"■";
50260 IF RV AND (NOT AN) THEN PRINT#1,"■";
50270 PRINT#1,CHR$(AS);:IF AS=34 THEN AN = (NOT AN)
50280 NEXT SP,ZL
50290 CLOSE1:SLOW
50300 RETURN
```

Diese Routine schaltet während der Hardcopy den FAST-Modus ein. Die ZL-Schleife durchläuft alle Bildschirmzeilen, die SP-Schleife alle Spalten. In RV und RO steht, ob der Reversdruck an oder aus ist, in AN steht, ob sich der Drucker im Quote Mode befindet.
In Zeile 50230 steht eine Codewandlungsformel, wie wir sie in 3.3 als Funktion definiert haben. Da die Codewandlung hier nur einmal erforderlich ist, lohnt eine Funktionsdefinition (DEF FN) nicht.
In Zeile 50250 steht das <RVS OFF>-Steuerzeichen am Ende, das durch CHR$(18) ersetzt werden kann, das <RVS ON> in Zeile 50260 durch CHR$(146). In Zeile 50290 wird der FAST-Modus wieder ausgeschaltet, in Zeile 50300 erfolgt der Rücksprung ins Unterprogramm, denn diese Routine ist als Unterprogramm vorgesehen.
Die Optimierung der Routine auf einen bestimmten Druckertyp hin soll angesprochen werden, allerdings sind die vorgestellten Routinen nur für den MPS 801/803 (nicht MPS 802!) geschrieben. Aufgrund der Programmbeschreibungen müßten aber auch die Besitzer anderer Drucker, wenn sie sich mit ihrem Gerät auskennen, in der Lage sein, Änderungen vorzunehmen.
Wir wollen jetzt das Anführungszeichen über den Grafikmodus senden, weil dann der Quote Mode verhindert wird. Dadurch werden die Flags RV, RO und AN überflüssig. In folgendem Programm für den MPS 801/803 steht das Senden des Anführungszeichens im Bitmodus in Zeile 50180.
Beschreibung: Hardcopy mit Anführungszeichen im Bitmodus (MPS 801)
Filename: »hc-801.1(text40)«

```
50000 REM ******************************
50010 REM **                          **
50020 REM ** ROUTINE FUER HARDCOPIES **
50030 REM **                          **
50040 REM ** (40-ZEICHEN-TEXT-MODUS) **
50050 REM **                          **
50060 REM ** SPEZIELL FUER MPS 801/3 **
50070 REM **                          **
50080 REM ******************************
50090 :
50100 SA=0:BANK 15:IF PEEK(53272) AND 2 THEN SA=7
50110 FAST:OPEN 1,4,SA:PRINT#1
50120 FOR ZL = 0 TO 24
50130 PRINT#1
50140 FOR SP = 0 TO 39
50150 AD=1024+ZL*40+SP:CC=PEEK(AD):IF CC>127 THEN PRINT#1,CHR$(18);
50160 BC=CC AND 127
50170 AS=BC+64+64*(BC<64ANDBC>31)+32*(BC<96ANDBC>63)
50180 IF AS=34 THEN PRINT#1,CHR$(8);CHR$(128);CHR$(135);CHR$(128);CHR$(135);CHR$
(128);CHR$(128);CHR$(15);:AS=0
50190 PRINT#1,CHR$(AS);:IF BC>127 THEN PRINT#1,CHR$(146);
50200 NEXT SP,ZL
50210 CLOSE1:SLOW
50220 RETURN
```

In Zeile 50180 wird der ASCII-Code auf 0 gesetzt, da dieser Code am Drucker kein Zeichen hervorruft, was sonst in Zeile 50190 geschehen würde.
Zusätzlich wollen wir noch Programmkosmetik betreiben und einige Verfeinerungen vornehmen. So soll der aktuelle Zeichensatz berücksichtigt werden, und der Abstand zwischen zwei

Zeilen der Hardcopy soll mittels Steuercodes auf ein Minimum reduziert werden. Wieder die Lösung für den MPS 801/803:
Beschreibung: Hardcopy mit Erkennung des Zeichensatzes
Filename: »hc-801.2(text40)«

```
50000 REM ******************************
50010 REM **                          **
50020 REM ** ROUTINE FUER HARDCOPIES **
50030 REM **                          **
50040 REM ** (40-ZEICHEN-TEXT-MODUS) **
50050 REM **                          **
50060 REM ** SPEZIELL FUER MPS 801/3 **
50070 REM **                          **
50080 REM ******************************
50090 :
50100 SA=0:BANK 15:IF PEEK(53272) AND 2 THEN SA=7
50110 FAST:OPEN 1,4,SA:PRINT#1
50120 FOR ZL = 0 TO 24
50130 PRINT#1,CHR$(8);CHR$(13);CHR$(15);
50140 FOR SP = 0 TO 39
50150 AD=1024+ZL*40+SP:CC=PEEK(AD):IF CC>127 THEN PRINT#1,CHR$(18);
50160 BC=CC AND 127
50170 AS=BC+64+64*(BC<64ANDBC>31)+32*(BC<96ANDBC>63)
50180 IF AS=34 THEN PRINT#1,CHR$(8);CHR$(128);CHR$(135);CHR$(128);CHR$(135);CHR$
(128);CHR$(128);CHR$(15);:AS=0
50190 PRINT#1,CHR$(AS);:IF BC>127 THEN PRINT#1,CHR$(146);
50200 NEXT SP,ZL
50210 CLOSE1:SLOW
50220 RETURN
```

Das Anführungszeichen wird nach wie vor im Bitmodus gesendet (Zeile 50180).
Zuletzt soll auch eine Maschinenroutine vorgestellt werden. Diese befindet sich auf der Programmdiskette und wird mit

```
BLOAD"HARDCOPY40.$1300", ON BO
```

geladen und mit

```
BANK 15:SYS DEC("1300")
```

gestartet.
Beides gleichzeitig bewirkt folgender Befehl, welcher mit einer 1541-Floppy nicht verwendet werden darf:

```
BOOT"HARDCOPY40.$1300",ON B15
```

Diese Routine kommt ohne Steuerzeichen aus, ist sehr schnell (da in Maschinensprache programmiert) und sicher auf den allermeisten Druckern lauffähig. Allerdings werden Reversdarstellungen am Bildschirm nicht auf den Drucker übertragen, um Steuerzeichen zu vermeiden. Hier der Quelltext vom Assembler TOP-ASS:

Beschreibung: Hardcopy vom 40-Zeichen-Bildschirm (Quelltext)
Filename: »hardcopy40.src«

```
READY.
100  -.BASE $1300                              ; OBJEKTCODE AB $1300 ABLEGEN
110  -;
120  -; ****************************************
130  -; *                                      *
140  -; *   H A R D C O P Y - R O U T I N E    *
150  -; *                                      *
160  -; *    VOM 40-ZEICHEN-TEXTBILDSCHIRM     *
170  -; *                                      *
180  -; ****************************************
190  -; *                                      *
200  -; *  PROGRAMMIERT VON: FLORIAN MUELLER   *
210  -; *               MIT: TOP-ASS           *
220  -; *                                      *
230  -; ****************************************
240  -;
250  -; SYMBOLDEFINITIONEN:
260  -;
270  -.DEFINE SETPAR              = $FFBA ; FILEPARAMETER SETZEN
280  -.DEFINE SETNAM              = $FFBD ; FILENAMEN SETZEN
290  -.DEFINE OPEN                = $FFC0 ; FILE OEFFNEN
300  -.DEFINE CLOSE               = $FFC3 ; FILE SCHLIESSEN
310  -.DEFINE BASOUT              = $FFD2 ; ZEICHEN AUSGEBEN
320  -.DEFINE CKOUT               = $FFC9 ; AUSGABE AUF FILE UMLENKEN
330  -.DEFINE CLRCH               = $FFCC ; AUSGABE WIEDER NORMAL
340  -;
350  -;
360  -.DEFINE ZEIGER              = $FB   ; ZEIGER AUF AKTUELLE
370  -.DEFINE ZEIGERLOW           = ZEIGER; POSITION IM
380  -.DEFINE ZEIGERHIGH          = ZEIGER+1; VDC-RAM
390  -.DEFINE ZEILE               = $FD
400  -.DEFINE SPALTE              = $FE
410  -;
420  -;
430  -                       LDA #<(1024)
440  -                       STA ZEIGERLOW
450  -                       LDA #>(1024)
460  -                       STA ZEIGERHIGH
470  -;
480  -                       LDA #0
490  -                       STA ZEILE
500  -                       STA SPALTE
510  -;
520  -                       LDA #4
530  -                       TAX
540  -                       LDY #0
550  -                       JSR SETPAR
560  -                       LDA #0
570  -                       JSR SETNAM
580  -                       JSR OPEN
590  -                       LDX #4
600  -                       JSR CKOUT
610  -;
620  -;
630  -SCHLEIFE               LDY #0
640  -;
650  -                       LDA SPALTE
660  -                       BNE WEITER1
670  -                       LDA #13
```

```
680  -                   JSR BASOUT
690  -;
700  -WEITER1            LDA (ZEIGER),Y
710  -;
720  -UMWANDLUNG         AND #%01111111
730  -                   TAX
740  -;
750  -                   CLC
760  -                   ADC #64
770  -;
780  -                   CMP #128
790  -                   BCS UMWANDLUNG1
800  -                   CMP #96
810  -                   BCC UMWANDLUNG1
820  -                   SBC #64
830  -;
840  -UMWANDLUNG1        CPX #96
850  -                   BCS AUSGABE
860  -                   CPX #64
870  -                   BCC AUSGABE
880  -                   SBC #32
890  -;
900  -AUSGABE            JSR BASOUT
910  -;
920  -                   BCS ENDE
930  -;
940  -                   INC ZEIGERLOW
950  -                   BNE WEITER2
960  -                   INC ZEIGERHIGH
970  -;
980  -WEITER2            INC SPALTE
990  -                   LDA SPALTE
1000 -                   CMP #40
1010 -                   BNE SCHLEIFE
1020 -                   LDA #0
1030 -                   STA SPALTE
1040 -                   INC ZEILE
1050 -                   LDA ZEILE
1060 -                   CMP #25
1070 -                   BNE SCHLEIFE
1080 -;
1090 -ENDE               LDA #13
1100 -                   JSR BASOUT
1110 -;
1120 -                   LDA #4
1130 -                   JSR CLOSE
1140 -;
1150 -                   JSR CLRCH
1160 -;
1170 -                   RTS
```

Hardcopy vom Textbildschirm im 80-Zeichen-Modus

Eine Hardcopy-Routine für den 80-Zeichen-Modus ist folgende:
Beschreibung: Hardcopy vom 80-Zeichen-Bildschirm
Filename: »hardcopy(text80)«

```
50000 REM ******************************
50010 REM **                          **
50020 REM ** ROUTINE FUER HARDCOPIES **
50030 REM **                          **
50040 REM **   VOM TEXT-BILDSCHIRM    **
50050 REM **                          **
50060 REM **   IM 80-ZEICHEN-MODUS    **
50070 REM **                          **
50080 REM ******************************
50090 :
50100 FAST:BANK 15:OPEN 4,4:PRINT#4
50110 FOR I = 0 TO 1999
50120 :SYS DEC("CDCC"),I/256,18
50130 :SYS DEC("CDCC"),I AND 255,19
50140 :SYS DEC("CDDA"),,31:RREG C:C=C AND 127
50150 :C=C+64+64*(C<64 AND C>31)+32*(C<96 AND C>63)
50160 :PRINT#4,CHR$(C);
50170 NEXT I
50180 PRINT#4:CLOSE 4
```

Diese Routine ist für (fast) alle Drucker verwendbar. Die Funktionsweise können Sie sich nach dem Abschnitt über VDC-Programmierung selbst erklären. Hier sei nur gesagt, daß jetzt nicht mehr jede Zeile mit <RETURN> beendet wird, da auch am Drucker eine Zeile 80 Zeichen hat. Reversdarstellung und Unterstreichen werden nicht berücksichtigt.
Eine entsprechende Routine in Maschinensprache laden Sie über

BLOAD"HARDCOPY80.$1300"

und starten Sie mit

BANK15:SYSDEC("1300")

oder Sie verwenden

BOOT »HARDCOPY80.$1300«,ON B15

für beides.

Hier der Quelltext für Maschinensprache-Interessierte:
Beschreibung: Hardcopy vom 80-Zeichen-Bildschirm (Quelltext)
Filename: »hardcopy80.src«

```
READY.
100  -.BASE $1300                        ; OBJEKTCODE AB $1300 ABLEGEN
110  -;
120  -; **************************************
130  -; *                                    *
140  -; *   H A R D C O P Y - R O U T I N E  *
150  -; *                                    *
160  -; *    VOM 80-ZEICHEN-TEXTBILDSCHIRM   *
170  -; *                                    *
180  -; **************************************
190  -; *                                    *
200  -; *  PROGRAMMIERT VON: FLORIAN MUELLER *
210  -; *                MIT: TOP-ASS        *
220  -; *                                    *
230  -; **************************************
240  -;
250  -; SYMBOLDEFINITIONEN:
260  -;
270  -.DEFINE SETPAR              = $FFBA ; FILEPARAMETER SETZEN
280  -.DEFINE SETNAM              = $FFBD ; FILENAMEN SETZEN
290  -.DEFINE OPEN                = $FFC0 ; FILE OEFFNEN
300  -.DEFINE CLOSE               = $FFC3 ; FILE SCHLIESSEN
310  -.DEFINE BASOUT              = $FFD2 ; ZEICHEN AUSGEBEN
320  -.DEFINE CKOUT               = $FFC9 ; AUSGABE AUF FILE UMLENKEN
330  -.DEFINE CLRCH               = $FFCC ; AUSGABE WIEDER NORMAL
340  -;
350  -.DEFINE SETVDC              = $CDCC ; VDC-REGISTER SETZEN
360  -.DEFINE GETVDC              = $CDDA ; VDC-REGISTER AUSLESEN
370  -;
380  -;
390  -.DEFINE ZEIGER              = $FB   ; ZEIGER AUF AKTUELLE
400  -.DEFINE ZEIGERLOW           = ZEIGER; POSITION IM
410  -.DEFINE ZEIGERHIGH          = ZEIGER+1; VDC-RAM
420  -;
430  -;
440  -;
450  -                  LDA #0         ; BEI $0000
460  -                  STA ZEIGERLOW  ; IM VDC-RAM
470  -                  STA ZEIGERHIGH ; BEGINNEN
480  -;
490  -                  LDA #4         ; FILENUMMER 4
500  -                  TAX            ; GERAETENUMMER AUCH 4
510  -                  LDY #0         ; SEKUNDAERADRESSE 0
520  -                  JSR SETPAR     ; FILEPARAMETER SETZEN
530  -;
540  -                  LDA #0         ; LAENGE = 0
550  -                  JSR SETNAM     ; ALSO KEIN FILENAME
560  -;
570  -                  JSR OPEN       ; FILE OEFFNEN
580  -;
590  -                  LDX #4         ; AUSGABE AUF FILE
600  -                  JSR CKOUT      ; NUMMER 4 UMLENKEN
610  -;
620  -SCHLEIFE          LDA ZEIGERHIGH ; HIGH-BYTE
630  -                  LDX #18        ; IN VDC-REGISTER 18
640  -                  JSR SETVDC     ; SCHREIBEN
650  -                  LDA ZEIGERLOW  ; LOW-BYTE
```

```
660  -                  LDX #19         ; IN VDC-REGISTER 19
670  -                  JSR SETVDC      ; (*** SIEHE BESCHREIBUNG !!! ***)
680  -                  LDX #31         ; VDC-REGISTER 31
690  -                  JSR GETVDC      ; AUSLESEN
700  -;
710  -UMWANDLUNG        AND #%01111111  ; BIT 7 AUSBLENDEN
720  -                  TAX             ; WERT IN X-REGISTER MERKEN
730  -;
740  -                  CLC             ; CARRY VOR ADDITION LOESCHEN
750  -                  ADC #64         ; 64 ADDIEREN
760  -;
770  -                  CMP #128        ; CODE < 128 ?
780  -                  BCS UMWANDLUNG1; NEIN, DANN NICHT 64 ABZIEHEN
790  -                  CMP #96         ; CODE >= 96 ?
800  -                  BCC UMWANDLUNG1; NEIN, DANN NICHT 64 ABZIEHEN
810  -                  SBC #64         ; 64 ABZIEHEN (BEI CODE<128 UND >=96)
820  -;
830  -UMWANDLUNG1       CPX #96         ; ALTER WERT (S. 720) < 96 ?
840  -                  BCS UMWANDLUNGENDE; NEIN, DANN NICHT 32 ABZIEHEN
850  -                  CPX #64         ; ALTER WERT (S. 720) >=64 ?
860  -                  BCC UMWANDLUNGENDE; NEIN, DANN NICHT 32 ABZIEHEN
870  -                  SBC #32         ; 32 ABZIEHEN (BEI 64<=ALTER WERT<96)
880  -;
890  -UMWANDLUNGENDE    JSR BASOUT      ; ZEICHEN NACH CODEWANDLUNG AUSGEBEN
900  -;
910  -                  BCS ENDE        ; EIN/AUSGABE-FEHLER (C=1)?
920  -;
930  -                  INC ZEIGERLOW   ; 16-BIT-ZEIGER AUF NAECHSTE
940  -                  BNE TEST        ; ADRESSE IM VDC-RAM
950  -                  INC ZEIGERHIGH  ; UM 1 ERHOEHEN
960  -;
970  -TEST              LDA ZEIGERLOW   ; NUN WIRD DER ZEIGER
980  -                  CMP #<(2000)    ; MIT 2000 VERGLICHEN
990  -                  LDA ZEIGERHIGH  ; BEI 2000: SCHLEIFENENDE
1000 -                  SBC #>(2000)    ; ANSONSTEN: SCHLEIFE FORTSETZEN
1010 -                  BCC SCHLEIFE    ;            MIT NAECHSTEM WERT
1020 -;
1030 -ENDE              LDA #13         ; CR (WAGENRUECKLAUF)
1040 -                  JSR BASOUT      ; AUSGEBEN
1050 -;
1060 -                  LDA #4          ; FILE NUMMER 4
1070 -                  JSR CLOSE       ; SCHLIESSEN
1080 -;
1090 -                  JSR CLRCH       ; AUSGABEGERAET WIEDER BILDSCHIRM
1100 -;
1110 -                  RTS             ; ROUTINE BEENDEN
```

Hardcopy vom Grafikbildschirm

Dies ist am schwierigsten, da jeder Drucker eine andere Hardcopy-Routine benötigt. Folgende Lösung gibt es aber:

1) Grafik auf Diskette im 1541-Format abspeichern:

   BSAVE "GRAPHIC",ON B0,P8192 TO P16383

2) In C64-Modus gehen (Reset + <CBM>)
3) Programm HI-EDDI PLUS laden
4) Grafikbild über <CBM>+<L> laden
5) Hardcopy mit <CBM>+<P>

Das Programm HI-EDDI PLUS, dessen Vorversion HI-EDDI in der Ausgabe 1/85 von 64er erschien, verfügt über eine umfangreiche Druckeranpassung. HI-EDDI PLUS wird mit umfangreicher Beschreibung von Markt & Technik vertrieben (Nummer MT 90136).
Da Simon's Basic eine Hardcopy-Routine für MPS 801/803 hat, soll hier auch eine Lösung für diesen Druckertyp vorgestellt werden, damit Sie durch diese Routine in Simon's-Basic-Programmen den Befehl COPY ersetzen können:
Beschreibung: Hardcopy vom Grafikbildschirm für MPS 801/803
Filename: »hardcopy(grafik)«

```
100 REM *********************************
110 REM *                               *
120 REM * H A R D C O P Y MPS 801/803   *
130 REM *                               *
140 REM *   VOM GRAPHIC-1-BILDSCHIRM    *
150 REM *                               *
160 REM *********************************
170 :
180 OPEN4,4:Z=0:CMD4
190 FORA=0TO27
200 FORB=0TO31
210 FORC=0TO10
220 FORD=0TO6:LOCATE(B*10+C),(A*7+D):IFRDOT(2)=1THENZ=Z+2↑D
230 NEXTD
240 Z=Z+128:A$=A$+CHR$(Z):Z=0:NEXTC
250 PRINTCHR$(8)A$;:A$="":NEXTB
260 PRINT" ":NEXTA
270 PRINT#4,CHR$(15):CLOSE4
```

Zusätzlich finden MPS 801/803-Besitzer noch folgende Programme auf der Programmdiskette:

"hardcopy-g.$1300"

Wird wie die anderen Hardcopy-Routinen in Maschinensprache gehandhabt

(BOOT"HARDCOPY-G.$1300"

oder BLOAD und SYS).

"hardcopy-g.src"

Der Quelltext zu »hardcopy-g.$1300«

"super-hc801$1300"

Gibt eine Hardcopy im Riesenformat aus und wird wie die anderen Hardcopy-Routinen in Maschinensprache verwendet.

3.8.3 Grafikbereiche schnell löschen

Da mit SCNCLR immer der ganze Bildschirm gelöscht wird, ist es schwer, nur einzelne Bereiche zu löschen. Wenn man aber mit dem CHAR-Befehl (in einer Schleife) entsprechend viele Leerzeichen sendet, ist dies leicht möglich. Das gilt sowohl für den Textmodus als auch die hochauflösende Grafik. Ein Beispiel finden Sie im Programm »Roulette« (3.6.2).

3.8.4 Spritesteuerung auf einfache Weise

Folgendes »Programm« ermöglicht eine einfache (aber schnelle) Joystickabfrage:

```
10 SPRDEF:SPRITE 1,1
20 J=(JOY(2)-1)*45:IF J=-45 THEN MOVSPR 1,0#0:GOTO 20:
   ELSE:MOVSPR 1,J#7:GOTO 20
```

Statt IF...THEN-Anweisungen werden die Joystickrichtungen in Grad gewandelt und später von einem MOVSPR-Befehl verarbeitet.
Zum Programm:
In Zeile 10 wird das Sprite mit Hilfe des SPRDEF-Sprite-Editors editiert und mit SPRITE1,1 eingeschaltet.
In Zeile 20 wird in der Variablen J die Richtung des Joysticks in Grad abgelegt (8 Richtungen).
Ist J=-45 (Nullstellung), wird das Sprite gestoppt und das Programm beginnt wieder bei Zeile 20.
Ist J nicht -45, bewegt sich das Sprite in Richtung J, worauf erneut Zeile 20 angesprungen wird.

3.8.5 Noch ein paar PEEKs und POKEs

POKE	160,255	ersetzt: TI$=»000000«
POKE	2603,32	abgeschalteter Cursor im 80-Zeichen-Modus
POKE	2603,64	schnell blinkender Cursor im 80-Zeichen-Modus
POKE	241,X-1	ersetzt: COLOR 1,X
POKE	216,0	ersetzt: GRAPHIC 0,1
POKE	216,1	ersetzt: GRAPHIC 1,1
POKE	216,64	ersetzt: GRAPHIC 2,1
POKE	216,128	ersetzt: GRAPHIC 3,1
POKE	216,254	ersetzt: GRAPHIC 4,1
POKE	243,1	ersetzt: PRINT CHR$(18); (RVS ON)
POKE	243,0	ersetzt: PRINT CHR$(146); (RVS OFF)

Eine ganze Reihe hochinteressanter PEEK- und POKE-Anweisungen finden Sie außerdem in 3.5.4.

Programmschutz-POKEs

Die Tasten <RUN/STOP> und <RESTORE> schaltet man mit

```
POKE 808,98
```

ab. Dadurch wird die interne Uhr (TI,TI$) unbrauchbar.
Einen Reset verhindert man mit

```
BANK1:POKE 65528,3
```

Dann führt jeder Reset zum Absturz. Dieser POKE funktioniert selbstverständlich auch ohne vorheriges Abschalten von <RUN/STOP> und <RESTORE>, aber beide POKEs zusammen ergeben einen recht guten Schutz, an dem sich andere die Zähne ausbeißen können.
Die Tastenkombination <RUN/STOP>+<RESTORE> kann außer mit »POKE808,98« auch durch »POKE792,51:POKE793,255« abgeschaltet werden, die <RUN/STOP>-Taste allein funktioniert aber weiterhin.
Mit »POKE 774,102:POKE 775,224« wird ein Listschutz aktiviert, der über »POKE 774,81:POKE 775,81« wieder aufgehoben werden kann.

4 Maschinensprache

Dieses Kapitel wendet sich an diejenigen, die bereits über Maschinensprachekenntnisse verfügen und diese auf dem C128 einsetzen wollen. Grundlagen in der Assemblerprogrammierung werden hier also nicht vermittelt (sondern in »C128: Programmieren in Maschinensprache«, Markt & Technik, Nummer MT 90213).
Auf jeden Fall sollten Sie sich aber noch zusätzlich ein ROM-Listing zulegen, da dies ein wichtiges Hilfsmittel zur Maschinenprogrammierung ist. Ein komplettes ROM-Listing ist das Buch »C128 ROM-Listing« aus der Commodore-Sachbuchreihe, die von Markt & Technik vertrieben wird.
Da der C128 mit dem 8502 eine Weiterentwicklung des 6510 (C64-Prozessor) als CPU hat, ändert sich an den Maschinenspracheanweisungen und Opcodes nichts. Sogar die sogenannten »undefinierten Opcodes«, also diejenigen Codes, die zwar in keinem Assembler-Lehrbuch stehen, aber dennoch vorhanden sind und Wirkung haben, wurden fast ausnahmslos übernommen, ebenso wie der technische Fehler, daß »JMP ($45FF)« nicht an die in $45FF/$4600 abgelegte Adresse springt, sondern an das in $4500 und $45FF festgelegte Sprungziel.
Man muß also keine neue Maschinensprache erlernen, allerdings muß man sich mit den Eigenheiten des C128 und den veränderten Bedingungen vertraut machen, wobei Ihnen dieses Kapitel helfen soll.
Sie werden aber sehen, daß aufgrund der komfortablen Anweisungen des Basic 7.0 in vielen Fällen auf die Programmierung einer Routine in Assembler verzichtet werden kann, wenn diese beispielsweise als Unterprogramm für ein Basic-Programm eingesetzt werden soll. Andererseits stehen aber, da das C128-ROM wesentlich umfangreicher als das C64-ROM ist, einige weitere ROM-Routinen zur Verfügung, und mit TOP-ASS (Markt & Technik) ist ein Assembler für den C128 erhältlich, der alle C64-Assembler übertrifft, da er von den erweiterten Möglichkeiten des C128 ausgiebig Gebrauch macht.
In 4.1 werden die Hilfsmittel zur Assemblerprogrammierung besprochen, in 4.2 das Umschrei-

ben von C64-Routinen. Auch wenn Sie nicht vorhaben, C64-Programme an den C128 anzupassen, sollten Sie diesen Abschnitt lesen, denn Sie erfahren dort vor allem die Eigenheiten des C128 gegenüber dem C64 (Bank-Switching, neue ROM-Routinen usw.).
Zum Bank-Switching wird Ihnen eine kleine Routinensammlung vorgestellt, die das Umschalten zwischen den Banks deutlich besser unterstützt als die entsprechenden Programmsegmente des Betriebssystems und des Basic-Interpreters.

4.1 Die Assembler-Tools

Während man mit dem, nach dem Einschalten sofort verfügbaren Basic-Interpreter ohne Verzögerung in Basic programmieren kann, muß man zur Programmierung in Assembler erst Hilfsprogramme (Tools) haben, die die Eingabe von Assembler-Quelltexten erlauben.
Zum Austesten und Bearbeiten eines Programms ist ein Monitor erforderlich. Dieser ist beim C64 erst in den Speicher zu laden, beim C128 ist ein recht guter Monitor bereits ins Betriebssystem integriert.
Ferner benötigt man, wie schon angesprochen, ein ROM-Listing, damit man sich nicht im Programmdschungel von Betriebssystem und Basic-Interpreter hoffnungslos verläuft.

4.1.1 Der Monitor

Der integrierte Monitor kann auf verschiedene Arten gestartet werden:
1. Eingabe des auf <F8> liegenden Befehls MONITOR.
2. Ausführung des Assembler-Befehls BRK (Opcode: $00).
3. Reset bei gedrückter <RUN/STOP>-Taste.

Der Monitor hält sich an einen gewissen Standard für 65xx-Monitore und ähnelt von der Syntax her den meisten C64-Monitoren (PROFI-MON, HESMON, DEMON). Dem sehr guten Monitor SMON entspricht fast kein Befehl, da der SMON eigenwillige Befehlsbezeichnungen hat.
Gegenüber einem C64-Monitor ändert sich vor allem, daß Adreßangaben jetzt bis zu fünf Stellen lang sind; die erste Stelle wählt die Bank aus (0,1,E oder F). Führende Nullen (einschließlich Angabe von Bank 0) können aber weggelassen werden; so syntax-freundlich sind C64-Monitore in der Regel nicht!

Beispiele: M F4000 F5FDC Hex-Dump von $4000 bis $5FDC in Bank $F
M 0 5 Hex-Dump von $0000 bis $0005 in Bank 0

Durch Voranstellen von »$« werden Hexadezimalzahlen gekennzeichnet ($ kann entfallen), durch »+« Dezimalzahlen, durch »&« Oktalzahlen und durch »%« Binärzahlen.

Beispiele: M +8192 +16383
M &4555 &1777

Bei Verwendung anderer Zahlensysteme als des 16er-Systems kann die Speicherbank nicht ausgewählt werden; Bank 0 ist dafür voreingestellt.
Die Befehle werden im Anhang C des C128-Handbuches erklärt. Ein Befehl wird dabei jedoch nicht erwähnt: J ist ähnlich wie G, allerdings wird – J steht für JSR – bei RTS in den Monitor zurückgesprungen, was nach G nur bei BRK der Fall ist.
Ein kleiner Tip: Wenn ein Basic-Programm unterbrochen werden soll, dieses aber geschützt ist, kann man einen Reset bei gedrückter <RUN/STOP>-Taste auslösen, »X« eingeben und dann das Programm editieren. Voraussetzung ist, daß das Programm nicht reset-geschützt ist.

4.1.2 Einsatz eines C64-Assemblers

Solange man noch keinen C128-Assembler hat, kann man sich mit der Verwendung eines C64-Assemblers behelfen, den man im C64-Modus betreibt. Bei der Assemblierung gibt es zwei Möglichkeiten:

Assemblierung auf Diskette

Ein auf Diskette erzeugtes Objektcode-File kann mit BLOAD in den C128 eingelesen werden.

Assemblierung in den Speicher

Wenn Sie ein C64-Programm in den Speicher assemblieren lassen, müssen Sie wissen, daß der im C64-Modus generierte Objektcode später in Bank 0 steht. Interessant ist der Bereich $1300-$1BFF (User-RAM-Bereich), in dem man kürzere C128-Maschinenprogramme unterbringt. Dann löst man einen Reset in den C128-Modus aus; der RAM-Bereich $1300-$1BFF wird dabei nicht initialisiert, der Bereich $1C03-$FEFF bleibt ebenfalls erhalten. Das RAM im C64-Modus steht in Bank 0 des C128-Modus und wird beim Moduswechsel nicht gelöscht.
Es ist aber auf jeden Fall anzuraten, einen C128-Assembler zu erwerben; das Produkt TOP-ASS ist so leistungsfähig und preiswert zugleich, daß das umständliche Assemblieren im C64-Modus wirklich nur eine provisorische Lösung ist; auf Dauer zahlt es sich aus, daß im C128-Modus mehr Speicher für den Quelltext, Objektcode und Label zur Verfügung steht. Die zusätzlichen Funktionen des TOP-ASS helfen vor allem, die Programmentwicklung komfortabler zu gestalten und zu beschleunigen. Mit TOP-ASS können auch C64-Maschinenprogramme assembliert werden (bei Speichern des Objektcodes auf Diskette), wodurch der C128 zum professionellen C64-Entwicklungssystem werden kann.

4.2 Umschreiben von C64-Routinen

Mit BLOAD oder dem Monitorbefehl »L« können auch C64-Files in den C128 geladen und verarbeitet werden. Quelltexte müssen aber neu eingegeben werden, wenn diese vom C128-Assembler TOP-ASS editiert und/oder assembliert werden sollen. Deshalb sollte man beim Umschreiben einen C64-Assembler verwenden, den man im C64-Modus betreibt; in 4.1.2 wurde der Einsatz eines C64-Assemblers bereits erklärt.
Vor allem muß man einen geeigneten Bereich im C128-Speicher suchen. Für kurze Routinen kommt der RS232-Kassettenpuffer in $0B00-$0BFF in Frage, für etwas längere der Bereich $1300-$1BFF (User-RAM-Bereich) und längere Programme können in Bank 0 ab Adresse $1C00, in Bank 1 ab $0400 stehen. Im 80-Zeichen-Modus kann der 40-Zeichen-Bildschirmspeicher $0400-$07E7, bei reinen Maschinenprogrammen – also solchen, die nicht mit einem Basic-Programm kommunizieren, steht auch der Bereich $0800-$09FF zur Verfügung (Basic-Stack für FOR/NEXT, DO/LOOP und GOSUB).

4.2.1 Zugriffe auf Adressen

Die Anpassung von POKE- und PEEK-Anweisungen, die schon sehr maschinennah sind, wird in 3.5.4 besprochen. Dort werden zwar oft Basic-7.0-Befehle als Ersatz vorgeschlagen, aber mit Hilfe eines ROM-Listings können diese in Maschinensprache umgesetzt werden.
Es ist jetzt zu beachten, daß Adressen beim C128 nicht einfach über LDA, STA usw. angesprochen werden, sondern daß die richtige Speicherbank eingestellt sein muß. Das Bank-Switching wird in 4.2.4 besprochen.
Hier noch eine Zusammenstellung von (brauchbaren) Zeropage-Adressen, die bei C64 und C128 vorkommen und an gleicher oder leicht verschobener Position im Speicher liegen, als Tabelle 4.1:

Tabelle 4.1: *Kurzer Adressenvergleich der Zeropage von C64 und C128*

Adressen bei C64	Beschreibung	Adressen bei C128
$07-$2A	Basic-Hilfszeiger	$09-$2C
$2B/$2C	Zeiger auf Basic-Anfang	$2D/$2E
$39/$3A	Zeiger auf aktuelle Zeile	$3B/$3C
$3F-$60	Hilfszeiger	$41-$62
$61-$66	Fließkomma-Akkumulator FAC	$63-$68
$67	Flag bei Polynomberechnung	$69
$69-$6E	Fließkomma-Akkumulator ARG	$6B-$70
$70-$72	diverse Zeiger	$6F-$73
$FA-$FE	Speicher zur freien Verfügung	$FA-$FE

Die meisten für Ein-/Ausgabe-Zwecke wichtigen Adressen stimmen überein, allerdings haben diese Adressen beim C128 meist zusätzliche Funktionen und können deshalb nicht tabellarisch abgehandelt werden. Bei Verwendung der Betriebssystem-Routinen sind diese Adressen aber in der Regel ohne Bedeutung.

4.2.2 Aufrufe von Betriebssystemroutinen

Da man unmöglich alle Einzelheiten wie Bildschirmausgabe selber programmieren kann, bedient man sich der im ROM enthaltenen Programme (ROM-Routinen). Diese sind beim C128 nur in Bank 15 ($F) verfügbar.
Die Aufrufe von Kernal-Einsprüngen des C64 ($FF81-$FFF3) müssen nicht angepaßt werden, da diese beim C128 auch vorhanden sind. Allerdings hat der C128 noch weitere interessante Kernal-Einsprünge, die in 4.2.3 aufgeführt sind.
Hier eine Zusammenstellung von arithmetischen Routinen von C64 und C128, da es für diese beim C64 keinen Sprungverteiler nach Kernal-Art gibt, als Tabelle 4.2:

Tabelle 4.2: *Gegenüberstellung arithmetischer Routinen von C64 und C128*

Bedeutung in Kurzform	C64	C128
Positive 2-Byte-Integerzahl nach Fließkomma	$BC49	$8C75
Negative 2-Byte-Integerzahl nach Fließkomma	$BC44	$8C70
Fließkomma nach Integer	$BC9B	$8CC7
Fließkomma nach 2-Byte-Integer	$B7F7	$8815
FAC in String	$BDDD	$8E42
String ausgeben	$AB1E	$55E2
Integerzahl (X/A) ausgeben	$BDCD	$8E32
FAC := ARG+FAC	$B86A	$8848
FAC := ARG-FAC	$B853	$8831
FAC := ARG*FAC	$BA2B	$8A27
FAC := ARG/FAC	$BB12	$8B4C
FAC := ARĜFAC	$BF7B	$AF39
FAC := FAC*10	$BAE2	$8B17
FAC := FAC/10	$BAFE	$8B38
Sinusfunktion SIN	$E26B	$AF42
Cosinusfunktion COS	$E264	$AF3F
Wurzelfunktion SQR	$BF71	$AF30
Logarithmus-naturalis-Funktion LOG	$B9EA	$AF2A
Exponentialfunktion EXP	$BFED	$AF3C
FAC nach ARG kopieren	$BC0C	$AF6C
ARG nach FAC kopieren	$BBFC	$AF69

Bedeutung in Kurzform	C64	C128
FAC nach Adresse kopieren	$BBD4	$AF66
Konstante nach FAC kopieren	$BBA2	$AF63
FAC := FAC+Konstante	$B867	$AF18
FAC := FAC*Konstante	$BA28	$AF1E
Auf Komma im Basic-Text prüfen (CHKCOM)	$AEFD	$795C
1-Byte-Integer aus Basic-Text holen (GETBYT)	$B79E	$87F4
Numerischen Ausdruck auswerten (FRMNUM)	$AD8A	$77D7
2-Byte-Integer aus Basic-Text holen (GETADR)	s.unten	$880F
Adresse und 1-Byte-Integer aus Basic-Text holen	$B7EB	$8803
Pointer auf Variable holen	$B08B	$7AAF

Die Funktion »2-Byte-Integer holen« (C128: $880F) wird beim C64 durch zwei Aufrufe ersetzt:

```
JSR $AD8A ; numerischen Ausdruck auswerten (FRMNUM)
JSR $B7F7 ; ins Adressformat wandeln
```

Beim C128 ersetzt man diese zwei Aufrufe, wenn Sie unmittelbar nacheinander stehen, durch »JSR $880F«.

4.2.3 Erweiterte Möglichkeiten auf dem C128

Hier sollen einige ROM-Routinen vorgestellt werden, die beim C64 nicht vorhanden sind, aber Ihnen die Programmierung sicher erleichtern.
Anspruch auf Vollständigkeit kann natürlich nicht erhoben werden, denn hier sind nur die wichtigsten Einsprünge aufgeführt.

WORDOUT ($B89F)

Eine 2-Byte-Adresse, die sich im Akku (Low-Byte) und X-Register (High-Byte) befindet, wird in hexadezimalem Format mit einem Leerzeichen am Ende ausgegeben.

HEXSOUT ($B8A5)

Das Byte im Akku wird hexadezimal (zweistellig) mit einem Leerzeichen am Ende ausgegeben.

HEXBOUT ($B8C2)

Wie HEXSOUT ($B8A5), aber ohne Leerzeichen.

MAKEHEX ($B8D2)

Das Byte im Akku wird in zwei hexadezimale Ziffern umgewandelt, wobei nach der Rückkehr aus der MAKEHEX-Routine die erste Stelle (oberes Nibble) im Akku und die zweite Stelle (unteres Nibble) im X-Register als ASCII-Code steht. Eine Ausgabe findet noch nicht statt.

MHEXBCD ($BA0A)

Ein maximal 6stelliger Hexadezimalwert wird ins BCD-System (Binary Coded Decimal) umgewandelt. BCD bedeutet hierbei, daß die jeweils zwei Dezimalziffern in einem Byte kodiert werden, wobei in jedem Nibble eine Ziffer von 0 bis 9 steht.
Der zu wandelnde Wert muß in den Adressen $66-$68 liegen ($68 = MSB, $66=LSB). Danach steht das Ergebnis in $0AA0-$0AA3 ($0AA0=MSB, $0AA3=LSB).

UNPACKB ($BA5D)

Mit Hilfe dieser Routine können beliebige Werte auf dem aktuellen Ausgabekanal ausgegeben werden, die vorher nach $0AA0-$0AA3 transportiert wurden ($0AA0=MSB, $0AA3=LSB). Die Ausgabe erfolgt binär, oktal oder dezimal; bei dezimaler Ausgabe muß vorher die Umwandlung ins Dezimale mit MHEXBDC erfolgt sein.
Folgende Parameter müssen angegeben werden:

Akku = Flag für Unterdrückung führender Nullen
0 = Führende Nullen unterdrücken
1 = Führende Nullen ausgeben

X = Anzahl der auszugebenden Ziffern

Y = Zahlensystem: 0 = binär
2 = oktal
3 = dezimal

VIDINIT ($C000)

Diese Routine setzt nach dem Einschalten bzw. Reset die Grundeinstellung der Video-Controller VIC und VDC sowie der für den Editor wichtigen Adressen im Speicher.

DISPLAY ($C003)

Ausgabe des Akkus auf den Bildschirm, Attribut (40-Zeichen-Bildschirm: Farbe; 80-Zeichen-Bildschirm: Farbe und Attribute). Das Attribut gliedert sich wie folgt:

Bits 0-3: Farbe
Bit 4: 0=Blinken aus/1=Blinken an
Bit 5: 0=Unterstreichen aus/1=Unterstreichen an
Bit 6: 0=Normaldarstellung/1=Reversdarstellung
Bit 7: 0=Groß-/Grafik-Zeichensatz/1=Klein-/Groß-Zeichensatz

GETKEYB ($C006)

Wie $FFE4 für Tastatur (GET von Tastatur).

GETSCRN ($C009)

Zeichen vom Bildschirm holen und in ASCII-Format wandeln. Gelesen wird aus der Adresse, die durch die aktuelle Bildschirmposition($E0/$E1) und den Offset ($EC) angegeben wird. Danach enthält der Akku das Zeichen im ASCII-Code, die Adresse $F2 das Attributbyte.

SCRNOUT ($C00C)

Wie $FFD2 für Bildschirm (BASOUT auf Bildschirm).

SCRNORG ($C00F)

Mit dieser Routine können Informationen über die Bildschirmorganisation geholt werden:

X = Anzahl der Spalten im Bildschirm(fenster)
Y = Anzahl der Zeilen im Bildschirm(fenster)
A = Modus (40 oder 80), unabhängig von Window

CURSORP ($C018)

Ist das Carry-Flag gesetzt (SEC), wird die Cursorposition nach X (Zeile) und Y (Spalte) geholt. Beide sind immer relativ zur linken oberen Ecke des aktuellen Windows zu betrachten.
Ist das Carry-Flag gelöscht (CLC), wird der Cursor entsprechend gesetzt (X=Zeile, Y=Spalte). Auch hier sind X und Y relativ zur Window-Position zu sehen.

ESCHDLG ($C01E)

Ein ESC-Kommando, das im Akku als ASCII-Code enthalten sein muß, kann hierdurch ausgeführt werden.

FKEYSET ($C021)

Mit dieser Routine wird eine Funktionstaste belegt. Als Beispiel soll <F1> mit »Testbelegung« belegt werden, der in Bank 0 ab Adresse $1E00 steht. Hierzu wird zunächst einmal ein Zeiger in der Zeropage eingerichtet, sinnvollerweise an einer frei verfügbaren Stelle, z.B. den Adressen $FA-$FE. Belegt wird also

$FA mit $00, dem Low-Byte der Adresse $1E00
$FB mit $1E, dem High-Byte dieser Adresse
$FC mit $00, der Banknummer

Anschließend werden die Prozessorregister geladen, und zwar:

A mit $FA, der Zeigeradresse
X mit $01, der Nummer der Funktionstaste
Y mit $0C, der Länge des Textes »Testbelegung«

Nun kann die Routine aufgerufen werden (JSR $C021). Möglich sind die Tasten <F1> bis <F8>, <HELP> (gilt als <F10>) und <SHIFT>+<RUN/STOP> (gilt als <F9>).

SWAPSCR ($C02A)

Diese Funktion schaltet jeweils auf den anderen Bildschirm, also von 40 auf 80 Zeichen und umgekehrt. Entspricht ESC-X.

WINDDEF ($C02D)

Definiert ein Window. Vor dem Aufruf steht im Akku die Zeilennummer (0-24), im X-Register die Spalte (0-39 bzw. 0-79). Bei gesetztem Carry-Flag (SEC) wird die rechte untere Ecke gesetzt, bei gelöschtem (CLC) die linke obere Ecke.

BEEPSGN ($C98E)

Diese Routine löst das Klingelzeichen (<CONTROL>+<G>) aus und ist, im Gegensatz zu CHR$(7), vom Quote Mode unabhängig.

LOCKSCB ($C8A6)

Verriegelt <SHIFT>+<CBM>, wie <CONTROL>+<K>.

UNLOCKS ($C8AC)

Entriegelt <SHIFT>+<CBM>, wie <CONTROL>+<L>.

FULLSCR ($CA24)

Schaltet Window aus (<HOME><HOME>).

DELLINE ($CA52)

Löscht die aktuelle Cursorzeile (ESC-D).

CURSRON ($CD6F)

Schaltet den Cursor, unabhängig von 40- und 80-Zeichen-Modus, im aktuellen Bildschirm ein.

CRSROFF ($CD9F)

Macht CURSRON rückgängig, schaltet also den Cursor aus.

Weitere Kernal-Einsprünge, die der C64 nicht hat:

SETSPIO ($FF47)

Schaltet die schnelle Floppy-Routine ein, die nur mit den Laufwerken 1570 und 1571 verwendbar ist. Das Carry bestimmt, ob die Routine eingeschaltet (CLC) oder ausgeschaltet (SEC) werden soll.

CLSALDV ($FF4A)

Schließt alle Files auf der im Akku angegebenen Geräteadresse.

C64MODE ($FF4D)

Schaltet auf den C64-Modus um. Von dort kann nur mit Reset in den C128-Modus zurückgesprungen werden. Entspricht »GO64«.

DMAREQC ($FF50)

Diese Routine dient zum direkten Speicherzugriff (DMA = Direct Memory Access) durch externe Geräte und ist im Normalbetrieb des C128 nicht von Belang, soll aber der Vollständigkeit halber erwähnt werden.

BOOTDSK ($FF53)

Im Akku muß die Laufwerksnummer (nicht Gerätenummer) im ASCII-Code stehen (»0«= $30, »1«=$31). Dann wird geprüft, ob die eingelegte Diskette bootfähig ist. Ist dies der Fall, wird das Bootprogramm ausgeführt, andernfalls wird der Aufruf ignoriert.

PHOENIX ($FF56)

Dient zum Aufruf der zusätzlichen Funktionskarte, die im normalen C128 nicht enthalten ist. Wird die Routine hier aufgerufen, wird ein normales BOOT von Gerät 8, Laufwerk 0 durchgeführt.

LFNKUPS ($FF59)

Zu einer vorgegebenen logischen Dateinummer im Akkumulator werden die zugehörige Gerätenummer und Sekundäradresse ermittelt. Ist das Carry-Flag nach dem Aufruf gesetzt, ist diese Datei nicht geöffnet. Ist das Carry-Flag gelöscht, so enthält das X-Register die Geräteadresse, das Y-Register die Sekundäradresse und der Akkumulator - wie vor dem Aufruf - die logische Dateinummer.

SECKUPS ($FF5C)

Zu einer vorgegebenen Sekundäradresse im Y-Register werden die zugehörige logische Dateinummer und die Sekundäradresse gelesen. Das Carry-Flag zeigt an, ob ein entsprechender Eintrag in der Dateitabelle gefunden wurde. Ist das Carry-Flag gesetzt (SEC), ist diese Datei nicht geöffnet. Ansonsten enthält das X-Register die Geräteadresse, der Akku die logische Dateinummer und das Y-Register - wie vor dem Aufruf - die Sekundäradresse.

KNLSWAP ($FF5F)

Ruft die bereits beschriebene Routine SWAPSCR ($C02A) auf.

KINIT80 ($FF62)

Initialisiert den Zeichensatz des 80-Zeichen-Bildschirms.

KFKEYST ($FF65)

Ruft die bereits beschriebene Routine FKEYSET ($C021) auf.

SETBANK ($FF68)

Setzt die Bank für LOAD/SAVE/VERIFY. Das X-Register muß die MMU-Bitkombination enthalten und der Akku die Banknummer. Der über diese Routine eingestellte Wert wird in $C6 (Akku) und $C7 (X-Register) zwischengespeichert.

GETCONF ($FF6B)

Diese Routine sucht zu einer gegebenen Banknummer im X-Register die entsprechende Kombination aus der Tabelle und stellt sie im Akkumulator zur Verfügung. Die Kombination kann z.B. bei SETBANK ($FF68) verwendet werden.

KJSRFAR ($FF6E), KJMPFAR ($FF71), KIFETCH ($FF74), KISTASH ($FF77), KCOMPAR ($FF7A)

Diese Routinen dienen dem Bank-Switching (4.2.4).

PRIMM ($FF7D)

Mit Hilfe dieser Routine kann ein Text, der direkt auf den JSR-Befehl folgt, auf dem Bildschirm ausgegeben werden. Der Text kann beliebig lang sein und muß mit $00 enden. Anschließend wird die Verarbeitung mit dem auf den Text folgenden Befehl fortgesetzt.

4.2.4 Bank-Switching in Maschinensprache

In Basic stellt man die aktuelle Speicherbank über den Befehl BANK ein. Wie dies in Maschinensprache geht, erfahren Sie im Anhang B des C128-Handbuches. Dort finden Sie eine Vielzahl von Abbildungen und eine Erklärung der MMU-Register.
Das Kernal stellt einige Routinen zum Bank-Switching zur Verfügung, die wir an dieser Stelle besprechen wollen.

KJSRFAR ($FF6E)

Diese Routine ruft ein Unterprogramm auf, das sich in einer beliebigen Speicherbank befinden kann. Hierzu muß das X-Register die Banknummer enthalten und einige Speicherstellen die entsprechenden gewünschten Registerinhalte beim Eintritt in das Unterprogramm. Die Logik entspricht einem JSR-Befehl. Hier die Speicherstellen für Register:

\$0006	Akkumulator-Inhalt
\$0007	X-Register-Inhalt
\$0008	Y-Register-Inhalt

KJMPFAR (\$FF71)

Diese Routine ist wie KJSRFAR (\$FF6E) zu steuern, die Logik entspricht aber nicht einem JSR-, sondern einem JMP-Befehl.

KIFETCH (\$FF74)

Mit Hilfe dieser Routine kann ein Byte aus einer anderen Speicherbank als der, in der das Programm gerade läuft, gelesen werden. Hierzu muß das X-Register die Banknummer und der Akkumulator die Adresse eines Zeropage-Adressenpaars enthalten, in dem die zu lesende Adresse steht. Die Funktion entspricht somit in etwa einem »LDA (Adresse),y«-Befehl.
Im Y-Register steht der Offset, wie bei LDA(),Y.

KISTASH (\$FF77)

Mit Hilfe dieser Routine kann ein Byte in eine andere Speicherbank als die, in der das aktuelle Programm gerade läuft, geschrieben werden. Hierzu muß das X-Register die Banknummer und die Speicherstelle \$02B9 die Anfangsadresse eines Adressenpaars der Zeropage enthalten, in dem die zu schreibende Adresse steht. Diese Funktion entspricht somit in etwa einem »STA (Adresse),y«-Befehl.
Im Y-Register steht der Offset, wie bei STA(),Y.

KCOMPAR (\$FF7A)

Mit Hilfe dieser Routine kann ein Byte im Akkumulator mit einem Byte aus einer anderen Speicherbank als der, in der das Programm gerade läuft, verglichen werden. Hierzu muß das X-Register die Banknummer enthalten und die Speicherstelle \$02CD die Anfangsadresse eines Adressenpaars aus der Zeropage, in dem die zu vergleichende Adresse steht. Die Funktion entspricht somit in etwa einem »CMP (Adresse),y«-Befehl. Im Y-Register steht der Offset, wie bei CMP(),Y.
Die genannten Routinen stehen auch an anderer Stelle im Speicher, wohin von den in Klammern aufgeführten Kernal-Einsprüngen aus nur gesprungen wird (Tabelle 4.3):

Tabelle 4.3: *Kernal-Einsprünge und Adresse in »common area«*

\$02A2	KIFETCH	(\$FF74)
\$02AF	KISTASH	(\$FF77)
\$02BE	KCOMPAR	(\$FF7A)
\$02CD	KJSRFAR	(\$FF6E)
\$02E3	KJMPFAR	(\$FF71)

Bank-Switching mit Hilfe einer eigenen Routine

Durch das Bank-Switching wird die Assemblerprogrammierung oft sehr aufwendig und mühselig. Wer schon einmal versucht hat, aus Bank 1 in eine Kernal-Routine nach Bank 15 zu springen, wunderte sich sicher, denn danach befand sich der Computer in einem überaus undefinierbarem Zustand...
Um diesem Problem zu begegnen, müßte ein Programm her, daß vor Aufruf die gewünschte Konfiguration einschaltet und bei der Rückkehr die alte Speicherbelegung wiederherstellt. Das Programm müßte zusätzlich in der »common area« (einem Speicherbereich, der allen Banks gemeinsam ist) liegen. Es bietet sich der RS232-Eingabepuffer von $0C00 bis $0D00 an, wobei allerdings eine Initialisierung nötig ist, die auch diesen Bereich als »common area« deklariert, denn normalerweise reicht die »common area« nur bis $03FF.
Ein solches Programm ist auf der Programmdiskette unter dem Filenamen »SWITCH« abgespeichert und muß nach dem Laden initialisiert werden. Dies erfolgt durch »JSR $0C3E«. Findet der Interpreter während der Ausführung eines Basic-Programms eine USR(X)-Anweisung, so wird nach $F000 in Bank 0 gesprungen.
Das Programm dient dazu, Unterprogramme in Bank 15 aus jeder beliebigen Bank aufzurufen, und funktioniert so:

Nummer der Kernal-Routine in Adresse 996 ($03E4) speichern

Die Bedeutung der Routinen wurde teilweise in 4.2.3, teilweise hier in 4.2.4 erklärt. Hier nur die zu der Routinennummer zugehörigen Adressen:

1=$FF47, 2=$FF4A, 3=$FF4D, 4=$FF50, 5=$FF53, 6=$FF56, 7=$FF59,
8=$FF5C, 9=$FF5F, 10=$FF62, 11=$FF65, 12=$FF68, 13=$FF6B, 14=$FF6E,
15=$FF71, 16=$FF74, 17=$FF77, 18=$FF7A, 19=$FF7D, 20=$FF81, 21=$FF84,
22=$FF87, 23=$FF8A, 24=$FF8D, 25=$FF90, 26=$FF93, 27=$FF96, 28=$FF99,
29=$FF9C, 30=$FF9F, 31=$FFA2, 32=$FFA5, 33=$FFA8, 34=$FFAB, 35=$FFAE,
36=$FFB1, 37=$FFB4, 38=$FFB7, 39=$FFBA, 40=$FFBD, 41=$FFC0, 42=$FFC3,
43=$FFC6, 44=$FFC9, 45=$FFCC, 46=$FFCF, 47=$FFD2, 48=$FFD5, 49=$FFD8,
50=$FFDB, 51=$FFDE, 52=$FFE1, 53=$FFE4, 54=$FFE7, 55=$FFEA, 56=$FFED,
57=$FFF0, 58=$FFF3.

Bei JSRFAR/JMPFAR auf andere Adresse in Bank 15 ist auf folgende Weise vorzugehen:

1) Die anzuspringende Bank ist in $02 abzulegen
2) Die anzuspringende Adresse ist in $03/$04 bereitzustellen
3) Akku in $06, X-Register in $07 und Y-Register in $08
4) JSR $0C23 bei JSRFAR; JMP $0CD0 bei JMPFAR

Diese seltsame Art der Parameterübergabe entspricht den Originalroutinen KJMPFAR und KJSRFAR, die bereits erwähnt wurden. Es dürfte also keine großen Probleme bei der Umrüstung bereits bestehender Programme geben.
Hier noch der Quelltext, der mit dem Assembler TOP-ASS erstellt wurde und somit von allen glücklichen Besitzern dieses Programms weiterverwendet werden kann:

Beschreibung: Quelltext zur »SWITCH«-Routinensammlung
Filename: »switch.src«

```
READY.
100  -; ***********************************************************
110  -; *                                                         *
120  -; *  Q U E L L T E X T   Z U R   "S W I T C H"-R O U T I N E *
130  -; *  =====================================================   *
140  -; *                                                         *
150  -; *  ERSTELLT MIT DEM ASSEMBLER: "TOP-ASS" (MARKT & TECHNIK) *
160  -; *                                                         *
170  -; ***********************************************************
180  -;
190  -                     .DEFINE BANK= $0002
200  -                     .DEFINE PCHIGH= $0003
210  -                     .DEFINE PCLOW= $0004
220  -                     .DEFINE STATUSREGISTER= $0005
230  -                     .DEFINE AKKU= $0006
240  -                     .DEFINE XREGISTER= $0007
250  -                     .DEFINE YREGISTER= $0008
260  -                     .DEFINE VARTAB= $002F
270  -                     .DEFINE ARRAYTAB= $0031
280  -                     .DEFINE NEWJSRFAR= $02CD
290  -                     .DEFINE NEWJMPFAR= $02E3
300  -                     .DEFINE ROUTINENNUMMER= $03E4
310  -                     .DEFINE USRVECTOR= $1219
320  -                     .DEFINE RAMCONFIGURATION= $D506
330  -                     .DEFINE CONFIGURATION= $FF00
340  -                     .DEFINE PEEKPOKE= $FF50
350  -;
360  -                     .BASE $0C00
370  -;
380  -;
390  -; *** KERNAL *** (VERTEILER)
400  -; NUMMER D. ROUTINE IN $03E4 (996)
410  -; DANN "JSR $0C00"
420  -; AUS BELIEBIGER BANK
430  -;
440  -KERNAL              STA AKKU        ; REGISTER
450  -                     STX XREGISTER   ; IN ZEROPAGE
460  -                     STY YREGISTER   ; RETTEN
470  -                     LDA #15         ; BANK 15
480  -                     STA BANK        ; EINSTELLEN
490  -                     LDA #<($FF44)   ; ADRESSE
500  -                     STA PCLOW       ; DER SPRUNGTABELLE
510  -                     LDA #>($FF44)   ; IN ZEROPAGE
520  -                     STA PCHIGH      ; ABLEGEN
530  -                     LDA ROUTINENNUMMER
540  -                     CMP #20         ; MIT 20 VERGLEICHEN
550  -                     BCC KLEINER     ; C=0 HEISST "KLEINER"
560  -                     INC PCLOW       ; LOW-BYTE DER SPRUNGTABELLE ERHOEHEN
570  -KLEINER             ASL             ; AKKU MIT 2 MULTIPLIZIEREN
580  -                     ADC ROUTINENNUMMER; UND NUMMER DER ROUTINE ADDIEREN
590  -                     ADC PCLOW       ; LOW-BYTE DER SPRUNGTABELLE ADDIEREN
600  -                     STA PCLOW       ; WERT IN LOW-BYTE DER SPRUNGTABELLE
610  -;
620  -; *** JSRFAR ***
630  -; PARAMETER WIE NORMALE JSRFAR-ROUTINE
640  -; AUFRUF DURCH "JSR $0C00"
650  -; AUS BELIEBIGER BANK
660  -;
670  -JSRFAR              LDA CONFIGURATION; KONFIGURATION VOR AUFRUF HOLEN
```

```
680  -                  PHA              ; UND AUF STAPEL ABLEGEN (ZWECKS SICHE
RUNG)
690  -                  LDA #0           ; BANK 15
700  -                  STA CONFIGURATION; EINSTELLEN
710  -                  JSR NEWJSRFAR    ; NEUE JSRFAR-ROUTINE AUFRUFEN
720  -                  PLA              ; WIEDER DIE ALTE KONFIGURATION
730  -                  STA CONFIGURATION; VOM STAPEL HOLEN UND HERSTELLEN
740  -                  LDA STATUSREGISTER; STATUS HOLEN
750  -                  PHA              ; UND ZUNAECHST AUF STAPEL LEGEN
760  -                  LDA AKKU         ; REGISTER
770  -                  LDX XREGISTER    ; AUS ZEROPAGE
780  -                  LDY YREGISTER    ; HOLEN
790  -                  PLP              ; STATUS WIEDERHERSTELLEN
800  -                  RTS              ; ENDE DER ROUTINE
810  -;
820  -; *** INIT *** (INITIALISIERUNG)
830  -; VERGROESSERT "COMMON AREA"
840  -; AUF 4K, PASST BASIC-ZEIGER AN
850  -; UND VERSCHIEBT PEEK/POKE-ROUTINE
860  -;
870  -INIT              LDA #0           ; BANK 15
880  -                  STA CONFIGURATION; EINSTELLEN
890  -                  LDA RAMCONFIGURATION; COMMON AREA
900  -                  ORA #5           ; AUF 4K
910  -                  STA RAMCONFIGURATION; ERWEITERN
920  -                  LDA #0           ; 0 ALS LOW-BYTE
930  -                  STA VARTAB       ; FUER ZEIGER AUF VARIABLEN-
940  -                  STA ARRAYTAB     ; UND ARRAY-BEGINN IN BANK 1
950  -                  LDA #$10         ; $10 ALS HIGH-BYTE ($1000 = #4096)
960  -                  STA VARTAB+1     ; IN GLEICHE ZEIGER WIE OBEN
970  -                  STA ARRAYTAB+1   ; SCHREIBEN
980  -;
990  -; ** USR-VEKTOR INITIALISIEREN **
1000 -;
1010 -INITUSR           LDA #<(START)    ; USR-VEKTOR AUF
1020 -                  STA USRVECTOR    ; "START" STELLEN, VON WO
1030 -                  LDA #>(START)    ; AUS BEI "USR" NACH $0F000
1040 -                  STA USRVECTOR+1; GESPRUNGEN WIRD
1050 -                  LDA #$7F         ; BANK 1
1060 -                  STA CONFIGURATION; EINSTELLEN
1070 -                  LDY #$37         ; $37+1 = $38 BYTES UMKOPIEREN
1080 -LOOP              LDA NEWPEEK,Y    ; SCHLEIFE
1090 -                  STA PEEKPOKE,Y   ; ZUM UMKOPIEREN
1100 -                  DEY              ; DER PEEK- UND POKE-
1110 -                  BPL LOOP         ; ROUTINEN
1120 -                  RTS              ; ENDE
1130 -;
1140 -; *** BEI "USR" NACH $0F00 ***
1150 -;
1160 -START             LDA #$3F         ; BANK 0
1170 -                  STA CONFIGURATION; EINSTELLEN
1180 -                  JMP $F000        ; SPRUNG NACH $0F000
1190 -;
1200 -; *** PEEK AUS BANK 1 ***
1210 -; X-REGISTER = LO-BYTE DER ADRESSE
1220 -; Y-REGISTER = HI-BYTE DER ADRESSE
1230 -;
1240 -; NACH DEM AUFRUF DER ROUTINE
1250 -; STEHT DANN IM AKKU DER PEEK-WERT
1260 -;
1270 -PEEKBANK1         STA PEEKLOW+1    ; LDA-BEFEHLE
1280 -                  STA PEEKHIGH+1   ; (ZEROPAGEADRESSIERT)
1290 -                  INC PEEKHIGH+1   ; MODIFIZIEREN
```

```
1300 -                   LDA CONFIGURATION; KONFIGURATION AUF STAPEL
1310 -                   PHA               ; ZWECKS SICHERUNG ABLEGEN
1320 -                   LDA #$7E          ; BANK 1,ABER I/O-BEREICH EINGEBLENDET
1330 -                   STA CONFIGURATION; ALS KONFIGURATION SETZEN
1340 -PEEKLOW            LDA 0             ; 0 = DUMMY
1350 -                   STA $FF5A         ; NACH $FF5A SCHREIBEN
1360 -PEEKHIGH           LDA 0             ; 0 = DUMMY
1370 -                   STA $FF5B         ; NACH $FF5A+1 = $FF5B SCHREIBEN
1380 -                   JSR PEEKPOKE      ; NEUE PEEK/POKE-ROUTINE AUFRUFEN
1390 -                   STA ROUTINENNUMMER; ERGEBNIS MERKEN
1400 -                   PLA               ; KONFIGURATION VOM STAPEL HOLEN
1410 -                   STA CONFIGURATION; UND EINSTELLEN
1420 -                   LDA ROUTINENNUMMER; ERGEBNIS DES PEEK-BEFEHLS HOLEN
1430 -                   RTS               ; ENDE DER ROUTINE
1440 -;
1450 -; *** POKE IN BANK 1 ***
1460 -;
1470 -; X-REGISTER = LO-BYTE DER ADRESSE
1480 -; Y-REGISTER = HI-BYTE DER ADRESSE
1490 -; AKKU       = ABZULEGENDER WERT
1500 -;
1510 -; KEINE RUECKGABEPARAMETER
1520 -;
1530 -POKEBANK1          STA ROUTINENNUMMER; WERT ZWISCHENSPEICHERN
1540 -POKEZP             LDA #$FF          ; POKE IN ZEROPAGE-ADRESSE
1550 -                   STA POKELOW+1     ; LDA-BEFEHLE
1560 -                   STA POKEHIGH+1    ; ENTSPRECHEND
1570 -                   INC POKEHIGH+1    ; MODIFIZIEREN
1580 -                   LDA CONFIGURATION; KONFIGURATION HOLEN
1590 -                   PHA               ; UND AUF STAPEL SICHERN
1600 -                   LDA #$7E          ; BANK1, ABER I/O-BEREICH EINGEBLENDET
1610 -                   STA CONFIGURATION; ALS KONFIGURATION SETZEN
1620 -POKELOW            LDA 0             ; 0 = DUMMY
1630 -                   STA $FF7C         ; NACH $FF7C SCHREIBEN
1640 -POKEHIGH           LDA 0             ; 0 = DUMMY
1650 -                   STA $FF7D         ; NACH $FF7C+1 = $FF7D SCHREIBEN
1660 -                   LDA ROUTINENNUMMER; GEMERKTEN WERT HOLEN
1670 -                   JSR $FF6C         ; NEUE POKE-ROUTINE ANSPRINGEN
1680 -                   PLA               ; ALTE SPEICHERKONFIGURATION
1690 -                   STA CONFIGURATION; WIEDERHERSTELLEN
1700 -                   RTS               ; ENDE DER ROUTINE
1710 -;
1720 -; *** JMPFAR ***
1730 -; PARAMETER WIE BEI NORMALER
1740 -; JMPFAR-ROUTINE
1750 -; DANN "JSR $0CD0" AUS JEDER BANK
1760 -;
1770 -JMPFAR             LDA #0            ; BANK 15 EINSTELLEN
1780 -                   STA CONFIGURATION; ALS KONFIGURATION
1790 -                   JMP NEWJMPFAR     ; NEUE JMPFAR-ROUTINE ANSPRINGEN
1800 -;
1810 -; * NEUE PEEK-ROUTINE *
1820 -;
1830 -NEWPEEK            SEI               ; INTERRUPT ABSCHALTEN
1840 -                   LDA RAMCONFIGURATION; COMMON AREA
1850 -                   AND #$F3          ; AUSSCHALTEN
1860 -                   STA RAMCONFIGURATION; UND ERGEBNIS ALS KONFIGURATION
1870 -                   LDA $FFFF,Y       ; WERT HOLEN
1880 -                   STA $FFF0         ; UND MERKEN
1890 -                   LDA RAMCONFIGURATION; COMMON AREA
1900 -                   ORA #$04          ; WIEDER
1910 -                   STA RAMCONFIGURATION; ANSCHALTEN
1920 -                   LDA $FFF0         ; WERT WIEDER HOLEN
```

```
1930 -                  CLI             ; INTERRUPT WIEDER ZULASSEN
1940 -                  RTS             ; ENDE DER ROUTINE
1950 -;
1960 -; * NEUE POKE-ROUTINE *
1970 -;
1980 -NEWPOKE           STA $FFF0       ; WERT MERKEN
1990 -                  SEI             ; INTERRUPT ABSCHALTEN
2000 -                  LDA RAMCONFIGURATION; COMMON AREA
2010 -                  AND #$F3        ; AUSSCHALTEN
2020 -                  STA RAMCONFIGURATION; UND KONFIGURATION SETZEN
2030 -                  LDA $FFF0       ; WERT WIEDER HOLEN
2040 -                  STA $FFFF,Y     ; WERT SETZEN
2050 -                  LDA RAMCONFIGURATION; COMMON AREA
2060 -                  ORA #$04        ; WIEDER
2070 -                  STA RAMCONFIGURATION; EINSCHALTEN
2080 -                  CLI             ; INTERRUPT WIEDER ZULASSEN
2090 -                  RTS             ; ENDE DER ROUTINE
```

In den Zeilen 530-600 wird die zur Kernal-Routine gehörige Adresse ermittelt. Dabei wird die Routinennummer verdreifacht (ein JMP-Befehl benötigt 3 Bytes, Kernal-Einsprünge sind nur JMP-Anweisungen), indem zuerst mit ASL verdoppelt (Zeile 570) und dann durch Addition verdreifacht (Zeile 590) wird. Da bei $FF80, also zwischen den Routinen 19 und 20, ein BRK-Befehl steht, muß zusätzlich 1 addiert werden, wenn die Routinennummer größer als 19 ist. Dafür sind die Zeilen 540-560 zuständig.
Dieser Wert wird zum in PCLOW/PCHIGH stehenden Wert $FF44 (siehe auch Zeilen 490-520) addiert, um die endgültige Adresse zu erhalten, die von der im Speicher unmittelbar folgenden JSRFAR-Routine für Bank 15 angesprungen wird.
Ansonsten reichen die Kommentare zum Verständnis des Programms mit Sicherheit aus.

5 Der C64-Modus

Ein Thema, das alle Umsteiger vom C64 brennend interessiert, soll jetzt ausführlich behandelt werden: der C64-Modus.
Laut Werbung ist dieser zu 100 Prozent kompatibel zum »richtigen« C64, was aber - wie wir noch sehen werden - nicht stimmt. Dennoch ist die Möglichkeit, den C128 als C64 zu betreiben, sehr interessant. Aus den zusätzlichen Fähigkeiten des C64-Modus gegenüber dem Original-C64 werden wir noch Kapital schlagen und den C64-Modus beschleunigen, die zusätzlichen Tasten (<ESC>, <TAB>, Zahlentastatur usw.) abfragen, den deutschen Zeichensatz einschalten und andere Hilfsprogramme einsetzen, die sich auf der Programmdiskette befinden. Wir werden auch ein Testprogramm entwickeln, das feststellt, ob es sich um einen C128/C64-Modus oder den »echten« C64 handelt.
Zu allererst soll aber die Wirkung von GO64, des Befehls, der in den C64-Modus springt, besprochen werden.
Man kann auch ohne diesen in den C64-Modus gelangen, wenn man beim Einschalten oder einem Reset die <CBM>-Taste gedrückt hält. Aus dem C64-Modus kann man nicht per Software in den C128-Modus zurückgelangen, das ist mittlerweile erwiesen. Aber durch einen Reset ist dies schon möglich. Nach folgendem Befehl, der vor GO64 einzugeben ist, wird auch bei einem Reset im C64-Modus in diesen zurückgesprungen, was zur Folge hat, daß man nur noch durch Aus- und Wiedereinschalten in den C128-Modus kommt:

```
BANK 1:POKE 65528,77:POKE 65529,255:GO 64
```

5.1 Was geschieht bei »GO 64«?

Wenn der Befehl GO64 im Direktmodus eingegeben wird, erfolgt zuerst die von manchen Diskettenbefehlen bekannte Sicherheitsabfrage »ARE YOU SURE ?«, die zur Bestätigung mit <Y><RETURN>, zur Ablehnung mit <RETURN> zu beantworten ist. Innerhalb eines Programms erfolgt diese Abfrage nicht. Im Direktmodus kann man mit der Anweisung

```
BANK 15:SYS DEC("FF4D")
```

in den C64-Modus gelangen, ohne daß »ARE YOU SURE?« gefragt wird.
Dieser SYS entspricht in der Wirkung exakt dem GO64-Befehl, da in dieselbe Routine hinter der Sicherheitsabfrage eingesprungen wird, und ist schnell in Assembler übersetzt: JMP $FF4D. So arbeiten GO64 und der SYS: Die Adressen 0 und 1 (Ein-/Ausgabe-Port und Datenrichtungsregister) werden initialisiert, eine Umsteigeroutine (darauf kommen wir gleich zu sprechen) wird in die Adressen 2-8 (Zeropage) kopiert, die Taktfrequenz auf 1MHz gesenkt (SLOW-Modus) und dann die vorher verschobene Umsteigeroutine bei Adresse 2 gestartet. Diese besteht aus drei Maschinensprache-Befehlen, die im »Mode Configuration Register« der MMU, also Adresse $D505, auf C64-Betrieb schalten und dann in die Reset-Routine des C64-Betriebssystems zu dessen Initialisierung springen. Durch das Umschalten des MMU-Register wird nämlich folgendes bewirkt (hier die drei wichtigsten Punkte):

1. Bank 0 wird als einzige (!) Bank eingeschaltet
2. Die C64-ROM-Bausteine werden eingeblendet. Sie enthalten das Original-C64-Betriebssystem
3. Die MMU-Register werden ausgeblendet, was ein softwaremäßiges Umschalten zurück in den C128-Modus unmöglich macht

Da das Original-C64-Betriebssystem im C64-Modus aus Kompatibilitätsgründen arbeitet, werden im C64-Modus die speziellen Eigenschaften des C128 (Funktionstastenbelegung, 80-Zeichen-Modus, doppelte Taktfrequenz, Sondertasten wie <ESC>, Zahlentastatur, eigener Cursortastenblock) nicht genutzt. Es werden aber noch Programme vorgestellt, die beispielsweise die Zahlentastatur abfragen, worauf das C64-Betriebssystem nicht eingestellt ist.

5.2 Die Unterschiede zum »Original«-C64

Da, wie in 5.1 gezeigt wurde, der C128 alle C64-Bausteine (Prozessor, ROM, RAM, Sound-Chip usw.) enthält und diese im C64-Modus betrieben werden, müßte der C128 wirklich 100%ig kompatibel - um nicht zu sagen: identisch - sein.
Leider ist dies nicht so, denn der C64-Modus verfügt zwar nicht über weniger Eigenschaften als der Original-C64, aber über einige zusätzliche Möglichkeiten, die hier besprochen werden sollen. Als Beispiel sei der deutsche Zeichensatz (DIN-Modus) genannt, der auch im C64-Modus

verfügbar ist; wenn dieser versehentlich von einem C64-Programm eingeschaltet wird, so kann dadurch z.B. bei manchen Spielen der Bildschirmaufbau so erheblich gestört werden, daß das Programm praktisch nicht funktionsfähig ist. Ein anderes Problem ist die doppelte Prozessorgeschwindigkeit (FAST-Modus): Auch wenn Basic 2.0 keinen FAST-Befehl hat (das Betriebssystem und der Basic-Interpreter sind ja identisch, weil sich die ROMs nicht unterscheiden), kann dessen Wirkung hervorgerufen werden. Bekannterweise »verschwindet« dann der Bildschirm, und der Anwender verliert auf diese Weise die Kontrolle über das Programm.
Die genannten Situationen sind beileibe keine rein theoretischen Fälle, sondern bei C64-Software schon wiederholt aufgetreten. Wenn Sie jetzt ein Urteil über den Grad der Kompatibilität erwarten, so muß ich Sie enttäuschen. Da nach und nach die Programme, die im C64-Modus des C128 Schwierigkeiten machen, von den Herstellern angepaßt werden, kann nicht gesagt werden, wieviel Prozent der derzeit erhältlichen C64-Software nicht lauffähig ist. Auf jeden Fall sind es deutlich unter 5 Prozent, und die Zahl schrumpft immer mehr.
Seien Sie also nicht allzu verärgert, wenn gerade eines Ihrer gekauften Programme nicht funktioniert, denn »kompatibel« heißt nicht »identisch«.
Von Vollkompatibilität kann man wirklich sprechen, denn es handelt sich nur um einige wenige Programme aus der riesigen C64-Software-Lawine, die nicht verwendbar ist.
Im Vergleich dazu: Besitzer von »IBM-kompatiblen PCs« (PC=Personal-Computer) sind froh und glücklich, wenn auf einem Kompatiblen etwa 80% der IBM-PC-Software laufen, solange die wichtigsten Programme verfügbar sind.
Wenn übrigens ein Programm von Diskette nicht läuft, so liegt dies an der Floppy; in Kapitel 7 wird dieses Thema ausführlich behandelt.

5.2.1 Der deutsche Zeichensatz

In 2.3 haben wir schon besprochen, wie der deutsche Zeichensatz eingeschaltet wird: Durch Umschalten eines Bits der Adresse 1 (Prozessor-Port) wird an physikalisch gleicher Adresse (53248-57343) anstelle des ASCII-Zeichensatzes das DIN-Zeichenmuster eingeblendet. Dazu muß das Betriebssystem des C128 gar nicht erst die Taste <CAPS LOCK> abfragen, denn das Umschalten geschieht automatisch, da die entsprechende Leitung unmittelbar mit <CAPS LOCK> verbunden ist.
Deshalb kann der DIN-Zeichensatz auch im C64-Modus mit <CAPS LOCK> oder den in 2.3 vorgestellten POKEs eingeschaltet werden. Im C64-Modus ist allerdings das C64-Betriebssystem nicht darauf eingerichtet (wie sollte es auch?) und kennt keine DIN-Tastaturbelegung. Deshalb liegen die deutschen Sonderzeichen auf ganz anderen Tasten als im C128-Modus. So liegt das Paragraphenzeichen jetzt auf <@> usw. Es gilt die »Commodore-128-Codetabelle« für den DIN-Modus, allerdings nicht für alle Steuerzeichen.
Wenn Sie die Taste feststellen wollen, die Sie betätigen müssen, so suchen Sie das gewünschte Zeichen in der DIN-Spalte der Tabelle und gehen Sie in der gleichen Zeile nach rechts in die ASCII-Spalte. Dort finden Sie dann das Zeichen, das bei Betätigung der entsprechenden Taste im ASCII-Modus erscheint. Anhand des Paragraphenzeichens und des Klammeraffen (<@>) können Sie dies nachvollziehen.

Die Möglichkeit, im C64-Modus den DIN-Zeichensatz darzustellen, hat aber auch eine andere Konsequenz. Wenn man beim C64 im Einschaltzustand »PRINT PEEK(1)« eingibt, so meldet er »55«. Im C64-Modus des C128 wird dabei nur im DIN-Modus »55« gemeldet, im normalerweise eingeschalteten ASCII-Zeichensatz ist das Ergebnis »119«. Allerdings führt »POKE 1,55« nicht gleich zum Einschalten des DIN-Modus, wurde aber vorher »POKE 0,255« ausgeführt, so wird mit »POKE 1,55« der DIN-Zeichensatz eingeschaltet.
Manche Programme fragen aus Kopierschutzgründen den Prozessor-Port (Adresse 1) ab und stürzen ab, wenn ein anderer Wert als 55 enthalten ist. Deshalb wird durch den deutschen Zeichensatz die Kompatibilität in zweifacher Hinsicht gefährdet: Die erste Möglichkeit ist die mißglückte Abfrage von Adresse 1, die zweite das unbeabsichtigte Einschalten des DIN-Modus.

5.2.2 Die Geisterbilder

Wie Sie aus C64-Zeiten wissen, kann mit »POKE 53280,Farbcode« die Rahmenfarbe geändert werden, was bei eingeschalteter Bank 15 auch im C128-Modus möglich ist. Die VIC-Register liegen im Bereich 53248-53296 (hexadezimal: $D000-$D030), und 53280 ist eines davon.
Beim C128 werden aber diese VIC-Register auch in den Adressen 53504-53552 ($D100-$D130), 53760-53808 ($D200-$D230) und 54016-54064 ($D300-$D330) eingeblendet, d.h. Sie können nach 53504 einen Wert POKEn, was exakt dieselbe Wirkung hat wie »POKE53248,-Wert«. Beim C64 zeigen POKEs in diese Kopien der VIC-Register keine Wirkung. Da die VIC-Register also mehrfach vorhanden zu sein scheinen (in Wirklichkeit handelt es sich immer um dieselben Register, die aber mit unterschiedlichen Adressen erreicht werden können), nennt man sie »Geisterbilder«. Es handelt sich dabei um eine C128-Spezialität (die auch im C64-Modus besteht), denn der Original-C64 bildet die VIC-Register nur an der Originaladresse 53248 ab.
Folgendes Beispiel schaltet die Rahmenfarbe auf Schwarz:

```
POKE 53280,0
```

Da die Geisterbilder alle 256 Bytes eingeblendet sind (nach 54016 allerdings nicht mehr), hat folgender POKE auf einem C128 auch im C64-Modus die Wirkung, daß der Rahmen weiß wird:

```
POKE 53280+256,1
```

Sogar folgende Möglichkeiten gibt es noch:

```
POKE 53280+2*256,1
POKE 53280+3*256,1
```

Auf einem echten C64 ist nur Adresse 53280 ansprechbar, die anderen POKEs werden ignoriert.

Übrigens können Werte in ein »Geisterbild« geschrieben, aber am Originalplatz (oder in einem anderen »Geisterbild«) ausgelesen werden:

```
POKE 53280+3*256,255:PRINT PEEK(53280)
```

ergibt 255.

5.2.3 Die zwei neuen VIC-Register

Gegenüber dem VIC des C64 sind beim VIC des C128, der auch im C64-Modus die Aufgaben des VIC übernimmt, zwei neue VIC-Register hinzugekommen.
Es handelt sich um die Register 47 (Adresse 53295) und 48 (Adresse 53296). Register 47 dient der Abfrage der Sondertasten (<ESC>, <TAB>, Zahlentastatur, Cursortastenblock usw.), Register 48 der Einstellung der Geschwindigkeit. Ist Bit 0 in Register 48 (Adresse 53296 = $D030) gelöscht, befindet sich der Prozessor im SLOW-Modus, was im C64-Modus vorgesehen ist, ist Bit 1 gesetzt, so läuft er mit doppelter Geschwindigkeit (FAST-Modus).
Schalten Sie zur Demonstration den FAST-Modus ein: POKE 53296,1
Erschrecken Sie nicht, wenn der Bildschirm jetzt furchtbar flackert; dies liegt daran, daß der VIC die doppelte Geschwindigkeit nicht mitmacht und verrückt spielt. Im C128-Modus ist dies genauso der Fall, auch dort können Sie diesen POKE eingeben. Der FAST-Befehl löst dies recht elegant, indem er den VIC einfach abschaltet, was ein weiterer POKE bewirkt. Erst versetzen Sie aber den C64-Modus in den Normalzustand (Reset), dann geben Sie folgende POKEs ein:

```
POKE 53296,1:POKE 53265,PEEK(53265) AND 239
```

Dann wird der FAST-Modus eingeschaltet, aber gleichzeitig wird der 40-Zeichen-Bildschirm verdeckt, damit das lästige Flimmern nicht zu sehen ist. Der FAST-Modus wird durch <RUN/STOP>+<RESTORE> nicht ausgeschaltet, da das C64-Betriebssystem die zusätzlichen VIC-Register nicht beeinflußt. Dazu sind folgende POKEs »blind« einzutippen:

```
POKE 53296,0:POKE 53265,PEEK(53265) OR 16
```

Durch die genannten Befehle, die übrigens exakt die Befehle FAST und SLOW des Basic 7.0 simulieren, kann man kritische Stellen eines Programms für den C64-Modus mit doppelter Geschwindigkeit ablaufen lassen. Dafür werden noch Hilfsprogramme vorgestellt, um beispielsweise durch Betätigen einer Tastenkombination den FAST-Modus einzuschalten.
Register 47 enthält in den Bits 0-2 Informationen über gedrückte Sondertasten. Daraus wird in 5.2.5 Kapital geschlagen.

5.2.4 C64 oder C128 im C64-Modus?

Um softwaremäßig zwischen Original-C64 und C64-Modus zu unterscheiden, gibt es sicher viele verschiedene Möglichkeiten, aber nur die zwei besten Lösungen sollen vorgestellt werden.

Geisterbilder

In 5.2.2 wurde schon erklärt, daß die VIC-Register beim C128 auch an anderer Adresse als der Originalposition 53248 ansprechbar sind. Folgende Eingabe gibt Aufschluß, ob es sich um einen C64 oder einen C128 handelt (die Kommentare sind nicht einzugeben):

```
POKE 53280,0:              Originalregister mit 0 belegen
POKE 53280+256,1:          Geisterbild des Registers auf 1 setzen
PRINT PEEK(53280)AND1      Originalregister auslesen
```

Das Ergebnis ist 0, wenn es sich um einen C64 handelt, 1 bei einem C128 im C64-Modus. Das Prinzip ist, daß zunächst das Originalregister mit dem Wert 0 (Farbcode »schwarz«) belegt wird. Dies funktioniert sowohl auf C64 als auch auf C128. Dann wird das »Geisterbild« dieses Registers angesprochen, was nur auf einem C128 möglich ist. Auf einem C128 bekommt das Register jetzt also den neuen Wert 1, auf einem C64 bleibt der zweite Befehl wirkungslos und somit der Wert 0 (erster POKE) erhalten. Zuletzt wird wieder das Register an Originalposition ausgelesen, was auf beiden Geräten möglich ist, aber aufgrund des Geisterbild-POKEs unterschiedliche Werte liefert.

FAST-Modus

Folgendes Basic-2.0-Programm basiert auf diesem Prinzip:
Beschreibung: Testprogramm »C64 oder C128?« mit FAST-Modus
Filename: »test 64/128«

```
10 REM ******************
20 REM *                *
30 REM *  TESTPROGRAMM  *
40 REM *                *
50 REM * C64 ODER C128? *
60 REM *                *
70 REM ******************
80 :
90 :
100 POKE 53296,1:POKE 53265,11:REM FAST-MODUS EIN
110 TI$="000000":REM UHR AUF 0 ZURUECKSETZEN
120 FOR F=1 TO 100:S=SIN(F)+COS(F)+TAN(F):NEXT
130 SEKUNDEN=INT(TI/60):REM ARBEITSDAUER ERRECHNEN
140 POKE 53296,0:POKE 53265,27:REM FAST-MODUS AUS
150 PRINT SEKUNDEN;"SEC. ARBEITSDAUER."
160 IF SEKUNDEN <10 THEN PRINT "C128 ODER C64-MODUS"
170 IF SEKUNDEN>=10 THEN PRINT "ORIGINAL C64"
```

In Zeile 100 versucht das Programm, den FAST-Modus einzuschalten, was nur auf einem C128 Wirkung hat. Dann wird die Zeit gemessen, die der Durchlauf eine Rechenschleife (Zeile 120) benötigt und wieder der FAST-Modus ausgeschaltet (Zeile 140). Auf einem C128 ist die Zeit nur halb so lang wie auf einem C64, wodurch das verwendete Gerät identifiziert werden kann (Zeilen 160-170).

5.2.5 Utilities zum C64-Modus

In Verbindung mit dem C64-Modus eröffnen sich Möglichkeiten, die auf einem »echten« C64 nicht bestehen. Diese sollen ausgenutzt werden, um die Arbeit mit dem C64-Modus zu erleichtern. Dadurch wird ein gewisser Ausgleich zu der durch die Zusatzfähigkeiten eingeschränkten Kompatibilität geschaffen werden.

Abfrage aller C128-Tasten: KEY 128

Mit dem Programm »KEY 128« können sämtliche C128-Tasten im C64-Modus benutzt werden, bis auf die Taste <CAPS LOCK>, deren Zustand in Adresse 1 steht, und <40/80 DISPLAY>, da diese Taste im C64-Modus nicht abgefragt werden kann (ihr Zustand liegt in einem MMU-Register, und diese werden beim Einschalten des C64-Modus bekanntlich ausgeblendet, siehe 5.1).
Das Programm wird im C64-Modus mit

```
LOAD"KEY 128",8
```

geladen und mit RUN gestartet.
Dann wird der FAST-Modus eingeschaltet, um das Einlesen der DATA-Zeilen zu beschleunigen, und die Werte im Bereich ab 49152 abgelegt. Anschließend werden zwei SYS-Befehle zum Ein- und Ausschalten der Abfrage der C128-Tastatur ausgegeben. Die Anfangsadresse des Maschinenprogramms im Speicher kann in Zeile 170 geändert werden. Manchmal kann es sein, daß bei einer bestimmten Zahl Schwierigkeiten auftreten, aber schon das Erhöhen der Startadresse um 1 alle Probleme beseitigt. Die vorbelegte Startadresse (49152) ist aber geprüft und garantiert fehlerfrei.
Ist die C128-Tastaturabfrage eingeschaltet, so können alle Tasten abgefragt werden, wenn man von <CAPS LOCK> und <40/80 DISPLAY> absieht, bei denen eine GET-Abfrage nicht möglich ist. Die separaten Cursor- bzw. Zahlentasten sind wie die entsprechenden Tasten auf der C64-Tastatur einzusetzen. Die Sondertasten wie <ESC> können zwar abgefragt werden, allerdings werden diese nicht wie im C128-Modus ausgeführt (ESC-Sequenzen stehen also nicht zur Verfügung, der C128 wird mit <NO SCROLL> nicht angehalten usw.).
Die Sondertasten müssen also ähnlich den C64-Funktionstasten abgefragt werden und liefern bei GET folgende ASCII-Codes, welche in Tabelle 5.1 aufgeführt sind:

Tabelle 5.1: *ASCII-Codes der Sondertasten*

3	<NO SCROLL>
8	<HELP>
9	<TAB> oder <SHIFT>+<HELP>
10	<LINE FEED>
14	<ALT>
24	<SHIFT>+<TAB>
27	<ESC>
142	<SHIFT>+<ALT>

Da die CHR$-Codes 8 und 9 im C64-Modus im Direktmodus die Kombination <SHIFT>+ <CBM> verbieten (8) oder zulassen (9), kann jetzt mit <HELP> diese Tastenkombination verhindert, mit <SHIFT>+<HELP> oder <TAB> wieder zugelassen werden.
Mit <ALT> kann aus gleichem Grund der Klein-/Großschrift-Modus aktiviert werden, mit <SHIFT>+<ALT> der Groß-/Grafik-Modus.
Auf der Programmdiskette befindet sich das im vorgesehenen Bereich (ab 49152) abgelegte Maschinenprogramm bereits als Objektcode und wird mit

```
LOAD"KEY.49152/49232",8,1
```

geladen. Darauf ist NEW einzugeben. Die C128-Tastaturabfrage wird mit SYS49152 ein-, mit SYS49232 ausgeschaltet.

C128 im C64-Modus um 35% schneller: FAST 64

Mit »FAST 64« wird der C128 im C64-Modus um 35% beschleunigt, ohne daß dadurch der Bildschirm ausgeblendet wird. Dies ist ein Kompromiß zwischen der FAST-Simulation mittels POKE, die zwar doppelte Geschwindigkeit erreicht, aber den Bildschirm abschaltet, und der langsamen Geschwindigkeit.
Dies wird dadurch erreicht, daß der 2MHz-Modus nur eingeschaltet wird, wenn der VIC den Rahmen ober- und unterhalb des Textes aufbaut und ohnehin nicht auf den Speicher zugreift, also im FAST-Modus betrieben werden kann. Dadurch ist der FAST-Modus zwar nicht immer eingeschaltet, was das Arbeitstempo verdoppelt, aber immerhin so oft, daß der C128 im C64-Modus um 35% schneller läuft.
Das Programm »FAST 64« wird wie »KEY 128« verwendet; die Startadresse steht jetzt allerdings in Zeile 160.
Auf Diskette ist noch »FAST.49440/49659«, also der Maschinencode, abgespeichert. Dieser wird wie »KEY.49152/49232« verwendet; der FAST-Modus wird mit SYS49440 ein-, mit SYS49659 ausgestellt.
Die Maschinencode-Files haben den Vorteil, daß man nicht auf das Einlesen der DATAs warten muß.
Bei eingeschalteter Beschleunigung dürfen keine I/O-Operationen mit externen Geräten (Floppy, Drucker) stattfinden.

»FAST 64« und »KEY 128« zusammen: KEY 128/FAST 64

Wenn man die beiden Programme »FAST 64« und »KEY 128« zusammen betreiben will, benötigt man das Programm »KEY 128/FAST 64«. Dieses liest beide Routinen in den Speicher ein und verbindet diese so, daß sie miteinander kooperieren.
Der Maschinencode ist unter der Bezeichnung »KEY/FAST64.OBJ« auf der Programmdiskette abgespeichert und wird wie die anderen Maschinencode-Files behandelt. Folgende SYS-Befehle gelten dann:

SYS 49152	»KEY 128« ein
SYS 49232	»KEY 128« aus
SYS 49440	»FAST 64« ein
SYS 49659	»FAST 64« aus
SYS 49152:SYS 49440	beide Programme ein
SYS 49232:SYS 49659	beide Programme aus

Der erste Doppel-SYS-Befehl ist unbedingt in der genannten Reihenfolge einzugeben; »SYS 49440:SYS 49152« würde zu Fehlfunktionen führen (beim Eingeben einer Sondertaste würde der obere Bildschirmrand flackern). Beim Ausschalten spielt die Reihenfolge allerdings keine Rolle.
Die Adressen können durch Ändern der Zuweisung »CODE=...« am Anfang des Programms »KEY 128/FAST 64« geändert werden.

Autochange für C128

Mit diesem Utility erkennt Ihr C128 automatisch beim BOOT-Versuch, welchen Modus - C64, C128 oder CP/M - er anspringen muß.
Den BOOT-Sektor wollen wir dahingehend ändern, daß er bei C64-Disketten automatisch die Kontrolle an das C64-Betriebssystem abgibt, wenn solche beim Einschalten des C128 oder bei einem Reset im Diskettenlaufwerk eingelegt sind. Folgendes Programm generiert einen BOOT-Sektor (Sektor 1,0) auf einer beliebigen im Commodore-Format beschriebenen Diskette, wodurch der C128 befähigt wird, eine C64-Diskette zu erkennen und automatisch nach dem Einschalten in den C64-Modus zu springen:
Beschreibung: Autochange für C128
Filename: »autochange«

```
100 REM *********************************
110 REM *                               *
120 REM * A U T O C H A N G E   128/64 *
130 REM *                               *
140 REM *********************************
150 REM *                               *
160 REM * BITTE IM C128-MODUS EINGEBEN *
170 REM *                               *
180 REM *********************************
190 :
200 IF RWINDOW(2)=80 THEN PRINT "PROGRAMM IST FUER 40-ZEICHEN-DARSTELLUNG VORGES
EHEN !":END
```

```
210 LC=0:COLOR0,16:COLOR4,7:COLOR5,1:SCNCLR
220 PRINT:PRINT"  DISKETTE EINLEGEN UND TASTE DRUECKEN"
230 PRINT"(D = DIRECTORY DER EINGELEGTEN DISKETTE)"
240 GETKEY X$:SCNCLR
250 IFX$="D" THEN DIRECTORY:GOTO 220
260 NA$=CHR$(147):FOR I=1 TO 7:NA$=NA$+CHR$(29):NEXT:FOR I=1 TO 7:NA$=NA$+CHR$(1
7):NEXT:NA$=NA$+"C64 DISKETTE"
270 CHAR,8,5,"AUTOMATISCHER WECHSEL"
280 CHAR,10,7,"IN DEN C64-MODUS"
290 OPEN15,8,15,"I0":A$=CHR$(0)
300 IFDS<>0 THEN BEGIN
310 CHAR,0,10,"DISKFEHLER: "+DS$,1
320 CLOSE8:CLOSE15
330 CHAR,11,15,"PROGRAMM-ABBRUCH":END
340 BEND
350 OPEN8,8,8,"#"
360 PRINT#15,"U1:8 0 18 0":PRINT#15,"B-P";8;5
370 GET#8,BA$
380 PRINT#15,"U1:8 0 1 0"
390 FOR I=0 TO 24:GET#8,S$:BL$=BL$+CHR$(ASC(S$)):NEXT
400 IF(ASC(BA$)AND1)=0 THEN BEGIN
410 IFLEFT$(BL$,3)="CBM"THENCHAR,5,10,"BOOT-SEKTOR SCHON VORHANDEN"
420 IFLEFT$(BL$,3)<>"CBM"THENCHAR,5,10,"PROGRAMM LIEGT AUF BOOT-SEKTOR"
430 CHAR,3,12,"INSTALLIERUNG FORTSETZEN? (J/N)":LC=1
440 GETKEY X$
450 IFX$<>"J"THENBEGIN
460 CLOSE8:CLOSE15
470 PRINT:END:BEND
480 BEND
490 PRINT:PRINT"----------------------------------------"
500 PRINTSPC(5)"INSTALLIEREN DES BOOT-SEKTORS"
510 PRINTSPC(4)"BITTE ENTFERNEN SIE EINEN EVTL."
520 PRINTSPC(5)"VORHANDENEN SCHREIBSCHUTZ UND"
530 PRINTSPC(8)"DRUECKEN SIE EINE TASTE"
540 GETKEY X$
550 PRINT#15,"B-P 8 0"
560 PRINT#8,CHR$(67);CHR$(66);CHR$(77);A$;A$;A$;A$;NA$;A$;
570 PRINT#8,A$;CHR$(32);CHR$(77);CHR$(255);A$
580 PRINT#15,"U2:";8;0;1;0
590 IFLC=0THENPRINT#15,"B-A 0 1 0"
600 CLOSE8:CLOSE15
610 PRINT:PRINT"BOOT-SEKTOR INSTALLIERT":END
```

Beim Erstellen dieses Befehlssektors werden keine eventuell vorhandenen Dateien auf der Diskette geschädigt, da dies gemeldet wird. Außerdem erkennt das Programm einen möglicherweise bereits installierten Bootsektor und weist, wenn vorhanden, darauf hin.
Bei der Warnung »Programm liegt auf BOOT-Sektor« sollte man den Bootsektor nicht installieren lassen, da sonst ein auf der Diskette befindliches Programm beschädigt wird.

Programme transferieren mit FLASHMOVE

Spielen Sie einmal folgenden Gedankengang durch: Sie besitzen einen C128 und eine Floppy 1570/1571. Diese Gerätekombination besitzt die gute Eigenschaft, daß im C128-Modus lange Programme in wenigen Sekunden geladen werden können. Sie möchten aber Ihre C64-Software aktiv weiterbenutzen. Also nichts wie die <CBM>-Taste beim Einschalten gedrückt halten und das entsprechende Programm laden ....

Man ärgert sich immer wieder, daß der C64-Modus längere Programme nur mit der gähnend langsamen 1541-Geschwindigkeit lädt. Es muß also eine Programmroutine her, die es ermöglicht, ein im C128-Modus mit höherer 1570/71-Ladegeschwindigkeit geladenes Programm in den C64-Modus zu holen, so daß man es dort wie gewohnt weiterverwenden kann. Die Lösung: »FLASHMOVE«.
Da Bank 0 sowohl im C128- als auch im C64-Modus der Speicherung von Programmen dient, der Basic-Anfang aber unterschiedlich ist (C64: 2048 = $0800; C128: 7168 = $1C00), liegt es nahe, daß ein im C128-Modus geladenes Programm nur noch verschoben werden muß, um es im C64-Modus (nach GO64) zu gebrauchen. Und eben das erledigt »FLASHMOVE GEN.«. Dieses Programm generiert unter einem einzugebenden Filenamen den Maschinencode, der auf der Programmdiskette unter der Bezeichnung »FLASHMOVE OBJ« steht.
Beschreibung: Flashmove-Maschinencode auf Diskette erzeugen
Filename: »flashmove gen.«

```
100 SCNCLR:PRINT"BITTE LEGEN SIE NUN EINE FORMATIERTE"
110 PRINT"DISKETTE IN DAS LAUFWERK!"
120 PRINT"DER MASCHINENCODE VON 'FLASHMOVE'"
130 PRINT"WIRD ABGESPEICHERT."
140 PRINT:POKE208,0:INPUT"FILENAME";F$
150 SCRATCH (F$)
160 OPEN1,8,1,F$:PRINT#1,CHR$(12)CHR$(8);
170 READA:IFA<>-1 THEN PRINT#1,CHR$(A);:X=X+A:GOTO170
180 CLOSE1:IFX<>13269 THEN PRINT:PRINT"PRUEFSUMMENFEHLER!!":END
190 PRINT:PRINT"DISKSTATUS: "DS$:END
200 DATA165,44,201,8,208,101,162,0,189,32,8,157,0,4,232,16,247,76,0,4,169,1
210 DATA141,32,208,173,17,18,133,2,56,233,20,133,46,173,16,18,133,45,120,169
220 DATA118,133,1,169,0,133,251,133,253,169,28,133,252,169,8,133,254,160,0
230 DATA177,251,145,253,200,208,249,165,252,197,2,240,6,230,252,230,254,208
240 DATA235,169,119,133,1,88,169,0,141,32,208,169,82,141,119,2,169,213,141
250 DATA120,2,169,2,133,198,76,51,165,96,-1
```

Der einzige Nachteil ist, daß das Programm nicht länger als 45K (das entspricht etwa 180 Blökken auf Diskette) sein darf. Übrigens, wenn die Anfangsadresse eines Programmes über 7168 ($1C00) liegt (beispielsweise ein Monitorprogramm ab 49152 = $C000), benötigt man Flashmove nicht. So benutzt man Flashmove:

1) Einschalten des C128-Modus
2) Flashmove mit BLOAD"FLASHMOVE OBJ" laden
3) DLOAD"C64-Programm"
4) GO 64 (bei GO64 wird Bank 0 nur teilweise gelöscht)
5) Maschinencode mit SYS2060 starten
6) Ende; das C64-Programm ist jetzt im C64-Modus verfügbar.

Der Blanker - Geschwindigkeit ist keine Hexerei

Wie Sie wissen, kann der C128 auch im C64-Modus im FAST-Modus (doppelte Taktfrequenz des Prozessors und dadurch doppelte Geschwindigkeit) betrieben werden. Wenn man bestehende Programme nicht umschreiben will, kann man mit dem Programm »BLANKER« bei Bedarf durch Drücken einer Tastenkombination von SLOW- auf FAST-Modus (oder umgekehrt) schalten.

Das Programm »BLANKER.LADER« auf der Programmdiskette legt den Maschinencode dieser kurzen Routine an einer angegebenen Startadresse im Speicher ab; wenn Sie bei der Eingabe nur <RETURN> drücken, wird 828 (Kassettenpuffer) als Startadresse angenommen. Es wird dann ein SYS-Befehl zum Aktivieren des Programms angegeben, der nur noch mit <RETURN> zu bestätigen ist.
Auf der Diskette ist eine Version von Blanker für den Bereich ab 49152 unter dem Filenamen »BLANKER $C000« abgespeichert, die wie die Maschinencode-Files bei den Utitilities a)-c) anzuwenden ist und mit SYS49152 aktiviert wird.
Mit <CONTROL>+<B> wird bei aktiviertem Blanker der FAST-Modus eingeschaltet. Mit derselben Tastenkombination wird er auch ausgeschaltet. Dabei sollte man zuerst <CONTROL> gedrückt halten, dann kurz <B> drücken (<CONTROL> bleibt gedrückt!) und dann sofort beide Tasten loslassen.
Bei eingeschalteter Beschleunigung dürfen keine I/O-Operationen mit externen Geräten (Floppy, Drucker) stattfinden.
Der Blanker wird durch <RUN/STOP>+<RESTORE> desaktiviert.
Hier der Quelltext (für Maschinensprachler), der mit dem C64-Assembler »HYPRA-ASS« (aus dem Sonderheft 8/85 des 64er-Magazins) erstellt wurde, da es sich um ein Programm für den C64-Modus handelt:
Beschreibung: Quelltext zu »BLANKER«
Filename: »blanker.src«

```
READY.
100  -; *** BLANKER - QUELLTEXT ***
110  -; ***     (HYPRA-ASS)     ***
120  -;
130  -;
140  -.EQ IRQVEC = $0314 ; IRQ-VEKTOR
150  -.EQ TASTE  =   $CB ; AKTUELLER TASTENCODE
160  -.EQ ALTIRQ = $EA31 ; ADRESSE DER ALTEN IRQ-ROUTINE
170  -.EQ BLANK1 = $D011 ; VIC-REGISTER #17
180  -.EQ BLANK2 = $D030 ; VIC-REGISTER #48
190  -.EQ CTRLFL = $028D ; FLAG FUER SHIFT,C=,CTRL
200  -.EQ ZAEHL  =   $B6 ; ZAEHLER
210  -;
220  -.BA $C000 ; AB 49152 ABLEGEN
230  -;
240  -;
250  -; INITIALISIERUNG DER NEUEN IRQ-ROUTINE
260  -;
270  -          SEI            ; INTERRUPT ABSCHALTEN
280  -;
290  -          LDA #<(NEUIRQ) ; IRQ-VEKTOR
300  -          LDY #>(NEUIRQ) ; AUF NEUE
310  -          STA IRQVEC     ; IRQ-ROUTINE
320  -          STY IRQVEC+1   ; STELLEN
330  -;
340  -          LDA #30
350  -          STA ZAEHL      ; ZAEHLER SETZEN
360  -;
370  -          CLI            ; INTERRUPT WIEDER EINSCHALTEN
380  -          RTS            ; ENDE DER INITIALISIERUNG
```

```
330  -;
400  -NEUIRQ    LDX ZAEHL
410  -          DEX
420  -          STX ZAEHL     ; ZAEHLER DEKREMENTIEREN
430  -          BMI PRUEF
440  -ENDEIRQ   JMP ALTIRQ    ; WEITER WIE BEI ALTEM IRQ
450  -;
460  -PRUEF     LDA #30
470  -          STA ZAEHL     ; ZAEHLER WIEDER SETZEN
480  -;
490  -          LDA TASTE
500  -          CMP #28       ; AUF "B" PRUEFEN
510  -          BNE ENDEIRQ   ; ENDE WENN <> "B"
520  -;
530  -          LDA #4        ; BIT FUER "CTRL" GESETZT
540  -          BIT CTRLFL    ; AUF CTRL-TASTE PRUEFEN
550  -          BEQ ENDEIRQ   ; NICHT CTRL GEDRUECKT, DANN ENDE
560  -;
570  -          LDA BLANK1
580  -          EOR #16       ; BLANK-BIT FLIPPEN
590  -          STA BLANK1    ; UND ABSPEICHERN
600  -;
610  -          LDA BLANK2
620  -          EOR #1        ; FAST-BIT INVERTIEREN
630  -          STA BLANK2    ; UND ABSPEICHERN
640  -;
650  -          JMP ALTIRQ    ; WEITER BEIM ALTEN IRQ

READY.
```

5.2.6 VDC-Programmierung im C64-Modus

Die Steueradressen 54784 ($D600) und 54785 ($D601) des VDC sind auch im C64-Modus ansprechbar, wenn auch das Betriebssystem keinen Gebrauch davon macht. Die Kompatibilität wird dadurch aber nicht eingeschränkt, da ein Beeinflussen des 80-Zeichen-Bildschirms die am 40er-Bildschirm laufenden C64-Programme nicht stört.
Man kann aber nach wie vor den VDC manipulieren. Allerdings ist es mit reinem POKEn in die entsprechenden Adressen nicht getan. Deshalb benötigt man Hilfsroutinen, wie sie im C128-Modus mit $CDCC und $CDDA zur Verfügung stehen. Deshalb hier das Assembler-Listing dieser C128-ROM-Routinen, die von Maschinenprogrammierern direkt für den C64-Modus übernommen werden können:

```
; Wert (Akku) in VDC-Register (X) schreiben
;
; im C128-ROM ab Adresse $CDCC in Bank $F
;
setreg    STX $D600     ; Nummer des Registers (X) setzen
waitset   BIT $D600     ; Bit 7 (Statusbit) des VDC prüfen
          BPL waitset   ; gelöscht (N=0), dann warten
          STA $D601     ; Akku (zu schreibenden Wert) übergeben
          RTS           ; Rücksprung von der Routine
```

```
;
;
;
; Wert aus VDC-Register (X) in Akku holen
;
; im C128-ROM ab Adresse $CDDA in Bank $F
;
getreg    STX $D600      ; Nummer des Registers (X) setzen
waitget   BIT $D600      ; Bit 7 (Statusbit) des VDC prüfen
          BPL waitget    ; gelöscht (N=0), dann warten
          LDA $D601      ; zu lesenden Wert in Akku holen
          RTS            ; Rücksprung von der Routine
```

Die Routinen unterscheiden sich im Prinzip nur im letzten Befehl vor RTS: Entweder wird ein Wert geschrieben, damit er in ein VDC-Register kommt, oder gelesen, damit er aus einem VDC-Register in den Akku geladen wird.

6 Einstieg in CP/M

Von den drei Betriebssystemen des C128 sind uns mittlerweile zwei geläufig: das C64- und das C128-Betriebssystem, welches dem C64-Betriebssystem in vielen Punkten entspricht.
Bleibt noch der Dritte im Bunde, nämlich CP/M. Dieses Betriebssystem unterscheidet sich in vielen wesentlichen Punkten vom C64-/C128-Modus, die wir nun besprechen wollen. Mit der Befehlserklärung möchte ich aber nicht zu weit gehen, denn im Handbuch finden Sie in Kapitel 7 eine komplette Beschreibung aller Editorfunktionen und Anweisungen.
Mit dem C128 können auch Commodore-Besitzer nun CP/M nutzen; das ist etwas, was bisher im Konzept dieses Herstellers gefehlt hat, läßt man einmal den Versuch außer acht, den C64 mit Hilfe eines Z80-Steckmoduls CP/M-fähig zu machen. Dies scheiterte zum einen am Bildschirmformat (nur 40 Zeichen sind für ein »professionelles Betriebssystem«, wie CP/M bezeichnet wird, zu wenig, aber der C64 schafft eben nicht 80 Zeichen pro Zeile) und zum anderen an der wenigen vorhandenen Software (die 1541 ist und war nicht in der Lage, verschiedene Diskettenformate zu lesen, aber außer den von Commodore mitgelieferten CP/M-Disketten war in diesem CP/M-unüblichen Format kein allzu breites Software-Angebot vorhanden).
Auf dem C128 hat sich dies grundlegend geändert: Der 80-Zeichen-Modus genügt professionellen Ansprüchen, und mittlerweile ist die gängigste Standardsoftware wie WordStar, dBase II, Multiplan, Turbo Pascal, CBASIC, MS-Basic auch im 1541/1570/1571-Format erhältlich. Dieses sensationell (man möchte fast sagen: branchenunüblich) preisgünstige Angebot hat übrigens zusätzlich dem CP/M-Modul für den C64 Auftrieb gegeben, denn vor kurzem wurden auch andere CP/M-Module von Fremdherstellern entwickelt.
Der C128 kann übrigens auch Programme verarbeiten, die mit dem CP/M-Modul für den C64 erstellt wurden, und CP/M-Disketten des C64 einlesen.
Dieses Kapitel soll keine komplette Einführung in CP/M sein, wie wie sie dem Kapitel 7 des Handbuches zu entnehmen ist, sondern vielmehr kurz erklären, welches Handwerkszeug der Anwender mit dem Betriebssystem CP/M auf dem C128 in die Hände bekommen hat.

6.1 Schnupperkurs zu CP/M

CP/M ist ein Betriebssystem und keine »Computersprache«, wie vielfach irrtümlicherweise angenommen wird. Übersetzt heißt es sinngemäß »Kontrollprogramm für Mikrocomputer«. Dieses Programm kontrolliert nun folgende Funktionen des Computers:

- Erkennen und Ausführen aller Eingaben
- Steuerung von Tastatur, Bildschirm, Drucker, Floppy etc.
- Komplette Verwaltung aller Daten auf Diskette
- Ausführung von Anwenderprogrammen

Das CP/M-System ist praktisch eine universelle Schnittstelle zwischen Anwenderprogramm und Hardware. Unter CP/M laufende Programm greifen daher nie direkt auf die Hardware zu (etwa durch POKEn eines Wertes in ein Register des Video-Controllers o.ä.), sondern wickeln alle Ein- und Ausgaben indirekt ab, nämlich durch Aufruf einer speziellen Betriebssystemroutine, dem sogenannten »BDOS CALL«. Maschinenprogrammierer kennen Vergleichbares vom C64/128-Kernal (Einsprünge ab $FF41 über zentralen Verteiler), allerdings ist CP/M hierbei noch viel konsequenter, da es wirklich alle Teile der Hardware steuert. Dagegen hat der C64/C128 keinen Kernal-Einsprung, um Bildschirmfarben zu setzen.

Der Aufbau von CP/M

Das eigentliche Betriebssystem besteht aus drei Hauptteilen, die beim »BOOTEN« des CP/M-Systems in den Speicher geladen werden und dort immer vorhanden (resident) sind. Dazu kommen eine Reihe »transienter« Kommandos, das sind Dienstprogramme, die nur bei Bedarf in den Speicher geladen werden.
Der speicherresidente Teil setzt sich zusammen aus dem BDOS (Basic Disk Operation System, »Basic« heißt »grundlegend« und steht nicht für die Programmiersprache), einem Grundsystem zur Diskettenverwaltung, dem BIOS (Basic Input Output System, auch hier hat »Basic« nichts mit der Programmiersprache zu tun), einem Anpassungsprogramm an die Hardware des Systems, und dem CCP (Console Command Processor), dem Kommando-Interpreter, der für die Ausführung von Benutzerkommandos verantwortlich ist. Wie Sie sehen, ist im Speicher zunächst also keine Programmiersprache oder dergleichen vorhanden; während man beim C64/C128 einfach in Basic »drauflos« programmieren kann, wenn das Gerät eingeschaltet ist, muß man unter CP/M zuerst die gewünschte Programmiersprache (Basic, Pascal, ADA, C, Forth, Cobol, Fortran, PL/1; unter CP/M ist fast alles zu haben!) einladen. Dies hat den Vorteil, daß man nicht auf die eingebaute Sprache (Basic beim C64/C128) festgelegt ist, aber den Nachteil, daß man eine Programmiersprache erst zusätzlich erwerben muß.
Um nun mit CP/M zu arbeiten, muß man das System BOOTen. Darunter wird das Kopieren des Programms von der Diskette in den Arbeitsspeicher verstanden. Dazu sollte man wie folgt vorgehen:

- C128 ausschalten
- mitgelieferte CP/M-Diskette einlegen

- Taste <CAPS LOCK> richtig einstellen (ASCII- oder DIN-Tastatur)
- Taste <40/80 DISPLAY> richtig einstellen (Bildschirmformat)
- C128 einschalten

Dann können Sie den Boot-Vorgang mitverfolgen. Es gibt noch weitere Möglichkeiten als die vorgestellte (siehe Handbuch), doch diese ist für CP/M-Anfänger am geeignetsten.
Noch ein Tip: Verwenden Sie unter CP/M nach Möglichkeit die vier separaten Cursortasten (also die grauen Cursortasten im abgesetzten Block), da die anderen manchmal nicht abgefragt werden (z.B. in Menüs) oder andere Funktionen haben.
Dieses BOOTen ist sicher für C64-Umsteiger ungewohnt, denn von C64- und C128-Modus kennt man, daß sie sofort nach dem Einschalten verfügbar sind. Aber die Methode, die Diskettenstation als zentrale Anlaufstelle zu benutzen, bietet einige Vorteile. So kann man, wenn man sich entsprechend auskennt, sehr leicht Hilfsprogramme als Kommandos einbinden.
Wenn CP/M fertig geBOOTet ist, werden Sie sehen, daß die Systemmeldung »A>« erscheint. »A« heißt, daß Diskettenlaufwerk A (Gerätenummer 8 in Normaleinstellung) angesprochen wird. CP/M spricht auch die Diskettenlaufwerke B, C und D an, wenn man diese anschließt und es mitteilt.
Im Speicher befindet sich nur ein minimaler Befehlssatz, die sogenannten »residenten Befehle«. Einer dieser Befehle ist DIR und ist vergleichbar mit dem Befehl DIRECTORY des Basic 7.0. Lassen Sie die CP/M-Systemdiskette unbedingt im Laufwerk und geben Sie jetzt DIR <RETURN> ein. Sie können auch auf <F3> drücken, da diese Taste unter CP/M mit diesem Text belegt ist. Es erscheint auf jeden Fall das Inhaltsverzeichnis der Systemdiskette. Wenn Sie einen 40-Zeichen-Bildschirm verwenden, wird mit <CONTROL>+<CRSR RIGHT> und <CONTROL>+<CRSR LEFT> (die separaten Cursortasten sind zu verwenden) der Bildschirm nach rechts und links gescrollt, da CP/M immer 80 Zeichen pro Zeile darstellt, von denen am 40er-Bildschirm nur die Hälfte sichtbar ist. Deshalb ist CP/M nur mit einem 80-Zeichen-Bildschirm praktikabel, was ja auch das CP/M-Modul für den C64 in seiner Verbreitung aufhielt.
Wie Sie jetzt sehen, unterscheidet sich ein CP/M-Directory erheblich von dem des C64/C128.
Als nächstes wollen wir eine Diskette formatieren (nehmen Sie dazu eine Leerdiskette oder eine, die Sie nicht mehr brauchen), denn CP/M kann nur unter CP/M formatierte Disketten lesen und beschreiben. Legen Sie aber zunächst die Systemdiskette ein und tippen Sie

```
FORMAT
```

ein, gefolgt von <RETURN> oder <ENTER>.
CP/M lädt dann dieses »transiente Kommando« von der Systemdiskette ein. Dieses arbeitet übrigens mit einer Menüsteuerung, wie wir sie in 3.6.5 programmiert haben, weshalb eine weitere Erklärung überflüssig ist (im Bedarfsfall können Sie im Handbuch nachlesen). Im Menü sind nur die Tasten aus dem separaten 4-Cursortasten-Block und <RETURN> verwendbar.
Nachdem wir jetzt zwei Befehle ausprobiert haben, wollen wir uns auch dem Editor zuwenden. Wie Sie gemerkt haben, ist es nicht möglich, den vollen Bildschirm zu editieren, denn <CRSR UP> und <CRSR DOWN> funktionieren gar nicht, <CRSR RIGHT> und <CRSR LEFT> nur innerhalb der Eingabezeile. Dies nennt man einen »Line Editor« (Zeilen-Editor), der C64 und C128 haben einen »Full Screen Editor« (Editor für den gesamten Bildschirm), der wesent-

lich höher entwickelt ist als ein »Line Editor«. Um aber die letzte Eingabe editieren zu können, kann man zwar nicht mit <CRSR UP> in die letzte Zeile fahren, aber immerhin mit <CONTROL>+<W> die letzte Eingabe in die aktuelle Eingabezeile holen. Statt <CONTROL>+<W> können Sie auch die <CRSR DOWN>-Taste im Schreibmaschinenblock (also rechts von <SHIFT>) einsetzen.
Innerhalb einer Eingabezeile stehen viele ESC- und CONTROL-Funktionen zur Verfügung (siehe Kapitel 7 des C128-Handbuches).
An dieser Stelle unterbrechen wir vorerst; wenn Sie sich weiterhin für CP/M interessieren, lesen Sie das entsprechende Kapitel im Handbuch. Hier finden Sie noch einige Tips zu CP/M.

Tips zu CP/M

- Wenn ein Fehler des Diskettenlaufwerks auftritt (LED blinkt), so kommt es vor, daß auch korrekte Disketten mit einer Fehlermeldung zurückgewiesen werden, da das Blinken nie aufhört. Abhilfe schafft hier das Aus- und Einschalten des Diskettenlaufwerkes, Besitzer eines 128D müssen den Floppy-Reset auslösen.
- Damit eine CP/M-Diskette als Systemdiskette erkannt wird, müssen sich die Files »CPM+.SYS« und »CCP.COM« auf ihr befinden. Die transienten Befehle sind nicht erforderlich, die wichtigsten sollte man aber kopieren.
  Diese dupliziert man mit Hilfe des (transienten) Kommandos PIP. Dieses wird im Handbuch beschrieben.
- Die Farben nach dem System-Boot (Rahmen ist hellbraun, Zeichen sind purpur) sind wahrlich nicht augenfreundlich. Es kann aber jeder seine Lieblingsfarbe oder die für die Augen angenehmste Farbkombination einstellen.
  Man ändert mit dem Programm KEYFIG.COM (starten über KEYFIG <RETURN>) einfach die Zuordnungen zwischen logischen und physikalischen Farben und speichert das Ganze ab. Nach dem nächsten Boot wird sofort die neue Farbbelegung aktiv. Hier die Zuordnungen:

  Rahmen: logische Farbe j
  Bildschirm: logische Farbe a
  Zeichen: logische Farbe e

  Diesen Farben kann man nun nach Belieben andere physikalische Farben zuordnen. Im einzelnen sieht dies folgendermaßen aus:
  Fertigen Sie als erstes eine Kopie Ihrer CP/M-Systemdiskette an (ganze Diskette kopieren!). Danach laden Sie KEYFIG (transientes Kommando) und wählen den zweiten Menüpunkt (»Definitions on the CP/M boot disk«) an. Nun wählen Sie wiederum den zweiten Menüpunkt (»Set up logical <-> physical colors«) und legen dann fest, ob sich die Farben auf den 40- oder 80-Zeichen-Bildschirm beziehen sollen. Nachdem Sie die Farben nach Wunsch eingestellt haben, speichern Sie die neuen Werte auf der Diskette (nicht auf der Original-CP/M-Systemdiskette!!).
  In den Menüs von CP/M können nur die Cursortasten im separaten Cursortastenblock eingesetzt werden.

- Die Möglichkeiten des Programmes KEYFIG gehen jedoch noch weit über das Ändern der Bildschirmfarben hinaus. Man kann auch Tasten mit anderen Funktionen belegen, beispielsweise auf <F1> die Spezialfunktion »Boot128«. Sobald man dann <F1> drückt, springt der C128 in den 128er-Modus und beginnt mit dem Booten (Systemreset). Da man die einzelnen Tasten auch mehrfach und mit Zeichenfolgen belegen kann, besteht durchaus die Möglichkeit, sich bei der Arbeit unter CP/M eine Menge Tipparbeit zu sparen. Zunächst aber soll ausführlich anhand des Beispiels »Boot128 auf<F1>« gezeigt werden, wie KEYFIG (befindet sich auf der Vorderseite der CP/M-Systemdiskette) angewendet wird.

  1) CP/M booten
  2) CP/M-Systemdiskette (Vorderseite) im Laufwerk lassen
  3) KEYFIG eingeben (startet KEYFIG.COM)
  4) Frage »Do you want help?« mit <N> verneinen
  5) Menüpunkt »Current definitions« anwählen
  6) Menüpunkt »Edit a key definition« anwählen
  7) <F1> drücken
  8) <RETURN> drücken
  9) Menüpunkt »ASSIGN a SPECIAL FUNCTION« anwählen
  10) Menüpunkt »Boot128« anwählen
  11) Menüpunkt »(done editing-exit and save work file« anwählen
  12) Menüpunkt »as CURRENT definitions« anwählen
  13) Frage »Do you want ...« mit <N> beantworten
  14) Sie befinden sich dann im Eingabemodus

- Da beim Verlassen von CP/M (über einen Reset oder die eben vorgestellte Methode mit »Boot128« auf<F1>) muß man die CP/M-Systemdiskette entfernen, da diese sonst automatisch erkannt wird und ein Booten auslöst.
- Mit Hilfe des transienten Kommandos

  ```
  DEVICE CONOUT:=40COL
  ```

  kann man die Ausgabe auf den 40-, mit

  ```
  DEVICE CONOUT:=80COL
  ```

  auf den 80-Zeichen-Bildschirm umlenken.
  Die Systemdiskette muß so im Laufwerk eingelegt sein, daß das Programm DEVICE.COM eingelesen werden kann, welches sich auf der Rückseite der CP/M-Systemdiskette befindet.
- Mit <CONTROL>+<C> kommt man aus vielen Hilfsprogrammen, die transiente Kommandos sind, zurück in den CP/M-Eingabemodus.
  <C> steht dabei für »Cancel« (engl. »annullieren«). Manchmal erfolgt die von manchen Basic-7.0-Befehlen her bekannte Abfrage »ARE YOU SURE ?«; wird diese mit <Y> (Yes) beantwortet, so wird in den Editor zurückgesprungen, bei <N> wird <CONTROL>+<C> ignoriert.
- Durch Eingabe von <CONTROL>+<Z> mit anschließendem <RETURN> wird der Bildschirm gelöscht, allerdings erscheint in der linken oberen Ecke ein Fragezeichen.

6.2 CP/M für Basic-Programmierer

CP/M selbst ist keine Programmiersprache, also wird man sich bald nach einer solchen umsehen, wenn man aktiv CP/M betreiben will. Dabei steht fast jede Programmiersprache zur Verfügung, aber auch zwei sehr interessante Basic-Implementierungen werden fertig angepaßt an den C128 angeboten, die für fortgeschrittene und professionelle Programmierer sehr interessant sind: CBasic von Digital Research und MS-Basic von Microsoft. Beide werden von Markt & Technik vertrieben.
Was soll man denn damit, wenn doch das leistungsfähige Basic 7.0 ohnehin eingebaut ist, werden Sie vielleicht fragen, und diese berechtigte Frage soll kurz beantwortet werden. Während Basic 7.0 als Weiterentwicklung des Basic 2.0 von C64/VC20 leicht zu erlernen ist (Kapitel 3 dieses Buches hat es gezeigt) und vor allem auf Grafik und Sound Wert legt, orientieren sich CBasic und MS-Basic an den Ansprüchen professioneller Programme und unterstützen somit vor allem die Datenverarbeitung. Wer »ernsthafte« CP/M-Software programmieren will, ohne sich erst mit dem Z80-Prozessor vertraut zu machen, wird mit diesen beiden Programmen den CP/M-Modus des C128 ausnutzen können, ohne eine völlig neue Programmiersprache lernen zu müssen, denn beide sind ja mit Basic 7.0 – mehr oder weniger – verwandt. Da jedes der beiden Programme seine eigenen Qualitäten hat, sollen diese getrennt vorgestellt werden.

CBasic – Mischung aus Basic und Pascal?

CBasic ist eine Programmiersprache, die zwar einen Befehlssatz wie Basic hat, aber für professionelle Programme vorgesehen ist; eine Reihe kommerzieller Anwendungen, vor allem für den kaufmännischen Bereich, wurde bereits in CBasic programmiert!
Damit das Endprodukt der Programmierung »professionell« ist, ist dieses Basic keine Interpretersprache wie Basic 2.0 und Basic 7.0, sondern eine Compilersprache wie Pascal, d.h. man gibt zwar Basic-Befehle (einen Quelltext) ein, dieser wird aber nicht mit RUN ausgeführt (interpretiert), sondern erst in die Z80-Maschinensprache übersetzt (compiliert). Da Maschinensprache wesentlich schneller als »höhere« Sprachen ist, bringt dies einen deutlichen Geschwindigkeitsvorteil. Allerdings ist die Fehlersuche etwas umständlicher, da man bei einer Fehlermeldung nicht den Objektcode (das vom Compiler generierte Maschinenprogramm) ändern kann, sondern nur den Quelltext, welcher zunächst in den Editor geladen werden muß. Compiler-Sprachen sind also teilweise anders ausgerichtet als Interpreter-Sprachen; da Pascal auch eine Compiler-Sprache ist, findet man viele compiler-typische Anweisungen (Arbeit mit Pointern usw.) auch in CBasic wieder. Der große Clou beim Compilieren ist aber, daß die Verständlichkeit der Sprache Basic mit der Geschwindigkeit von (schwer zu erlernender) Maschinensprache verbunden wird. Nebenbei verfügt der CBasic-Compiler über 14stellige Dezimalarithmetik, gewährleistet dabei höchste Genauigkeit bei Berechnungen und umgeht Rundungsfehler (geben Sie doch einmal im C128- oder C64-Modus »PRINT 3↑4« ein; CBasic würde das korrekte Ergebnis 81 liefern, aber Basic 2.0/7.0 ....). Wie bei Compilern üblich, wird die Integerarithmetik, die in Maschinensprache leicht und effektiv programmiert werden kann, unterstützt, was die Geschwindigkeit sehr erhöht. Die Möglichkeit, mehrzeilige Funktionen zu

erstellen, die »DEF FN« alt aussehen lassen und sonst nur in strukturierten Programmiersprachen wie C, Pascal oder PL/1 zu finden sind, und die Fähigkeit, innerhalb einer solchen Funktion lokale Variable zu verwenden, erhöhen die Struktur von CBasic-Programmen erheblich und übertreffen die DO-LOOP-Schleifen des Basic 7.0 bei weitem. Abgerundet wird dies durch eine umfangreiche Stringverarbeitung bei einer Stringlänge bis 32K, womit CBasic auch bei der Programmierung einer Textverarbeitung (!) eingesetzt werden kann.
Bei der Eingabe sind jetzt keine Zeilennummern mehr erforderlich, da Label gesetzt werden. Überhaupt ist die Eingabe eher mit einer Textverarbeitung als einem Editor wie im Commodore-Basic zu vergleichen.
Zusammenfassend kann gesagt werden, daß ein erfahrener Anwender, dem auch nicht mehr so viele Fehler (Bugs) bei der Programmierung unterlaufen, mit CBasic leistungsfähige Basic-Programme erstellen kann. Auch wenn CBasic an der oberen Grenze von Basic liegt und fast schon zu Sprachen wie C und Pascal gerechnet werden muß, ist es doch eine Programmiersprache, die auf dem Prinzip von Basic beruht und somit leicht erlernbar ist.
CBasic-Programme können sehr leicht auf andere CP/M-Computer (Schneider, IBM etc.) übertragen werden, meist ist nur eine Neu-Compilierung erforderlich. Man kann also mit CBasic einen weitaus größeren Bereich von Anwendern abdecken, als dies mit dem C128-Modus, der nicht so verbreitet ist wie CP/M, möglich wäre.

MS-Basic - Ein bewährter Oldtimer

Bereits 1975 entwickelte Microsoft den ersten Basic-Interpreter für Mikrocomputer, heute ist MS-Basic auf mehr als 1 000 000 Computersystemen installiert. Fast alle bekannten Mikrocomputer unterstützen Microsoft-Basic. Zahlreiche Anwendungsprogramme wurden von den verschiedenen Benutzergruppen und Softwareherstellern in MS-Basic geschrieben.
Wenn Sie Ihren C128 einschalten, erscheint die Einschaltmeldung, aus der auch hervorgeht, daß das Basic 7.0 des C128 - übrigens ebenso wie die anderen Basic-Versionen auf Commodore-Computern - eine Entwicklung von Microsoft ist. Dies macht sich auch in der Ähnlichkeit von MS-Basic und Basic 7.0 bemerkbar, die sich in grundlegenden Punkten gleichen. Vor allem die Art der Syntax ist ziemlich ähnlich. Allerdings ist MS-Basic wesentlich komfortabler als Basic 7.0 und ist eine der umfassendsten Implementationen der Sprache Basic, die derzeit auf 8-Bit-Mikrocomputern zur Verfügung steht. Auch die strukturierte Programmierung wird durch einige Anweisungen unterstützt, aber die Programme werden nach wie vor mit Zeilennummern eingegeben.
MS-Basic verfügt sowohl über einen Interpreter zum Austesten der Programme als auch über einen Compiler, der mit dem Interpreter bearbeitete Programme in 3- bis 10mal schnelleren Maschinencode übersetzt; sogar ein Assembler wird mitgeliefert, um den Objektcode (Maschinencode, der aus dem Basic-Quelltext erzeugt wurde) zu verändern.
Nicht nur Syntax, sondern auch Befehlssatz des Basic 7.0 findet man bei MS-Basic wieder, wobei MS-Basic weder Grafik noch Sound, aber dafür um so mehr die Dateiverwaltung, Stringbearbeitung und Bildschirmausgabe unterstützt. Die grundlegenden Befehle haben aber gleiche Syntax und sind in beiden Versionen vorhanden.

Vergleich zwischen CBasic und MS-Basic

Wenn man beide Basic-Implementierungen vergleicht, muß man feststellen, daß es sich um zwei hochkarätige Softwareprodukte handelt. Wer auf Grafik und Sound verzichten kann und sich mehr für Profi-Anwendungen interessiert, sollte nicht länger zögern und von Basic 7.0 auf eine der beiden anderen Basic-Versionen umsteigen. Die Profis werden sicher den CBasic-Compiler vorziehen, da dieser leistungsfähigere Programme erzeugt als MS-Basic, und für erfahrene Anwender mehr bietet, aber wer ein Basic sucht, daß mit Basic 2.0/7.0 verwandt und dennoch bei ernsthaften Anwendungen wertvoller ist, dürfte eher zu MS-Basic tendieren, wofür auch der vorhandene Interpreter, der das Austesten erheblich erleichtert, spricht.
Mehr Leistungsfähigkeit muß CBasic zugesprochen werden, was bei sehr erfahrenen Programmierern sicher den Ausschlag gibt, aber ein Umsteigen auf MS-Basic ist leichter, da

- Basic 7.0 und MS-Basic »blutsverwandt« sind
- jeder mit dem Interpreter auch mal »drauflos programmieren« kann, ohne daß jeder kleine Fehler ein Neuladen von Editor und Quelltext sowie ein erneutes Compilieren erfordert.

Legen Sie Wert auf Grafik und Sound und sind Sie bereits sehr vertraut mit Basic 2.0, so ist eine Alternative zum Basic 7.0 kaum nötig, da dieses in den genannten Bereichen alles bietet.

7 Die Diskettenlaufwerke zum C128

Der C128 kann mit der (mehr oder weniger guten) alten 1541 oder einer der beiden neuen Floppies 1570 und 1571 betrieben werden. Da sich diese Diskettenlaufwerke nicht nur in ihren Namen unterscheiden, sollen in diesem Kapitel die Unterschiede und Gemeinsamkeiten aufgegriffen werden.
Es sei noch gesagt, daß das im 128D eingebaute Laufwerk eines vom Typ 1571 ist und von der Softwareseite kein Unterschied zur »echten« 1571 besteht.
In 7.1-7.3 stelle ich Ihnen die 3 Laufwerke vor, in 7.4 wird die Kompatibilitätsfrage zwischen den Laufwerken geklärt und in 7.5 werden die Floppies verglichen, wobei Sie noch mit dem einen oder anderen Tip versorgt werden.

7.1 Die VC 1541

Das Standardlaufwerk zum C64 ist zweifellos die VC 1541, mag über sie noch so sehr (oftmals zu Recht) gelästert werden. Da dieses Gerät mit dem C64 eine weite Verbreitung fand und es am C128 auch verwendet werden kann, können sich viele C128er nicht durchringen, ein technisch neueres und besseres Laufwerk zu erwerben, wenn Sie bereits eine 1541 besitzen.
Beim Betrieb im C64-Modus ist die VC 1541 zweifellos am geeignetsten, denn für den C64 wurde sie ja auch entwickelt. Mit Hilfe eines Floppy-Speeders als Software-Lösung (z.B. HYPRA-LOAD, das Listing des Monats in der Ausgabe 10/1984 von 64'er) oder Hardware-Erweiterung kann man deren Geschwindigkeit auf ein erträgliches Niveau bringen, damit die langen Kaffeepausen der Vergangenheit angehören. Am besten ist aber die Anschaffung eines Floppy-Speeders, der auch im C128- und CP/M-Modus die 1541 beschleunigt. Zu vielen C64-

Floppy-Speedern sind aber Updates erschienen, um Klassiker unter den Speedern auch im C64-Mdus des C128 verwenden zu können.
Der C128 ist zwar in der Lage, die 1541 zu steuern, allerdings ist diese so langsam, daß im CP/M-Modus die Arbeit mit der 1541 kaum möglich ist. Es sei nur als Beispiel angeführt, daß allein das Booten mehr als zwei Minuten dauert. Wenn man dann noch bedenkt, daß die meisten Befehle transient sind, also von Diskette geladen werden müssen, summiert sich dies ins Unerträgliche. Auch der wenige, pro Diskette zur Verfügung stehende Speicherplatz von ca. 170K ist ein Problem, denn wenn ein Programm den Speicher des C128 voll ausnutzt, belegt dieser mit 128K etwa 3/4 einer Diskettenseite, von der Verwendung einer Speichererweiterung (RAM-Disk) ganz zu schweigen.
Allerdings haben die Softwarehersteller Rücksicht auf die Anwender der Floppy 1541 genommen, denn es gibt keine C128-Software, die nicht im 1541-Format ausgeliefert würde, weil dieses von allen drei Floppies gelesen werden kann.
Da die Floppy 1541 kein neues Gerät und sehr ausführlich dokumentiert ist (das Buch »Die Floppy 1541«, Markt & Technik, ist dabei ganz besonders hervorzuheben), kann ich Ihnen keinen neuen Tip geben. Die alte Warnung, den Klammeraffen (@) beim Abspeichern zu verwenden, um das alte File zu löschen, kann nur wiederholt werden, denn auch der DSAVE-Befehl kann diesen Fehler im DOS (Disk Operating System = Betriebssystem der Floppy) der 1541 nicht beheben. Statt

```
DSAVE"@PROGRAMM"
```

schreibt man einfach die fehlerfrei funktionierende Eingabe

```
SCRATCH"PROGRAMM":DSAVE"PROGRAMM"
```

So ist man wenigstens vor der Zerstörung einer Diskette sicher.
Ein Problem bei der Arbeit mit der 1541 ist der BOOT-Befehl des Basic 7.0, der aufgrund eines Betriebssystemfehlers (die Floppy ist schuldlos) nicht funktioniert. Zwar kann BOOT zum Booten einer BOOT-fähigen Diskette (CP/M-Systemdiskette etc.) verwendet werden, aber nicht zum Laden und Starten eines Maschinenprogramms.

```
BOOT"FILENAME"
```

ist also mit der 1541 nicht möglich, obwohl es mit der 1570/1571 problemlos arbeitet.
Ein Ausweg ist die Ermittlung der Anfangsadresse des Programms mit folgender Zeile:

```
0 OPEN 1,8,0,"NAME":GET#1,A$,B$:PRINT ASC(A$)+ASC(B$)*256:CLOSE 1
```

Nehmen wir an, diese Zeile ergibt den Wert 4864, so schreiben Sie danach statt BOOT»-NAME« folgenden Ersatz:

```
BLOAD"NAME":SYS 4864
```

In einem Programm kann man folgende allgemeingültige Zeile verwenden:

```
0 OPEN 1,8,0,"NAME":GET#1,A$,B$:SA=ASC(A$)+ASC(B$)*256:CLOSE1:
  BLOAD"NAME":SYS SA
```

Das Booten bei eingelegter BOOT-Diskette, das automatisch beim Einschalten oder Eingabe von BOOT (ohne Parameter) erfolgt, ist aber ohne solche Verrenkungen möglich. Allerdings wird das Booten automatisch bei Reset bzw. Einschalten ausgeführt, was zu einem »Rattern« bei der Positionierung des Schreib-/Lese-Kopfes führt (Fachausdruck: Bump). Dieses ist nicht nur akustisch gesehen unangenehm, sondern auch der Lebensdauer und Justierung der Floppy äußerst abträglich, weshalb man immer eine formatierte Diskette vor dem Einschalten des C128 einlegen sollte. Dies gilt nicht für die 1570 und 1571, welche über eine Lichtschranke verfügen, ist aber auch mit diesen beiden Geräten ratsam, damit der C128 schneller nach dem Einchalten verfügbar ist.
Man sollte nur darauf achten, daß es sich um keine CP/M-Diskette oder sonstige Disk handelt, die einen ungewollten Bootvorgang auslöst; am besten ist eine vom selben Laufwerk im C128-Modus formatierte Disk geeignet.

7.2 Die Floppy C 1570

Da es mit der neuen Floppy 1571 anfangs Produktionsschwierigkeiten gab, wurde als Ersatz das Laufwerk 1570 angeboten. Dieses ist, als Übergangslösung gedacht, eine Kreuzung aus dem schnellen DOS der 1571, das wesentliche Geschwindigkeitsvorteile gegenüber der 1541 hat, und der Hardware der 1541, da die Mechanik der 1571, die Disketten doppelseitig beschreiben kann, nicht lieferbar war.
Die 1570 war vor allem für diejenigen interessant, die unbedingt eine schnellere Floppy wollten, aber auf doppelte Diskettenkapazität, sprich die 1571, nicht warten wollten.
Auch die 1570 kann im C64-Modus verwendet werden, wie die 1541 und die 1571 ebenfalls.
Da sich außer der Behandlung doppelseitiger Disketten kein Unterschied zwischen 1570 und 1571 in der Behandlung ergibt, lesen Sie als 1570-Besitzer auch den Abschnitt über die 1571. Die dort vermittelten Tips beziehen sich auf beide Floppies. Generell gilt jedoch: Jeder Befehl, der sich auf zwei Diskettenseiten bezieht beziehungsweise die Wahl zwischen der einen (Seite 0) oder der anderen (Seite 1) der Diskette offenläßt, gilt nur für die 1571. Bei der 1570 muß dann immer Seite 0 gewählt werden; ansonsten ist eine Fehlermeldung der Floppy die Folge.

7.3 Die Floppy C 1571

Da dies das eigentlich vorgesehene Standardlaufwerk zum C128 ist, soll ihm besonders viel Raum gewidmet werden.
Eines der wichtigsten Verkaufsargumente für die 1570/71-Laufwerke ist ihr stark beschleunigter Busbetrieb. Zusammen mit dem C128 werden Geschwindigkeiten erreicht, von denen ein C64 mit 1541 ohne Beschleuniger bisher nur träumen konnte. Wer sich allerdings nun parallele Datenübertragung oder ähnliches vorstellt, muß leider enttäuscht werden. Der normale

serielle Bus des C64 wurde nur leicht umgebaut. Eine bisher unbenutzte Leitung wird jetzt zur Synchronisation der schnellen Datenübertragung genutzt.
Ein Vergleich überrascht in den ersten Augenblicken angenehm: Ein Testprogramm von 200 Blöcken auf der Diskette wird vom C64(-Modus) und der 1541 (ohne Speeder) in etwas über zwei Minuten geladen. Die 1570/71 schafft im C128-Modus dasselbe Programm in unter 14 Sekunden! Voraussetzung ist allerdings, daß das Programm auch im C128-Modus mit der 1570/71 gespeichert wurde. Bei vom C64 gespeicherten Programmen dieser Größenordnung werden 25 Sekunden benötigt, die Zeit, die auch von Speed-Dos, Turbo-Access oder Hypra-Load erreicht wird. Beim Speichern eben dieser 200 Blöcke erweist sich die 1570/71 als genauso langsam wie die 1541. Sie benötigt 1 Minute 52 Sekunden gegenüber 2 Minuten 2 Sekunden bei der 1541. Andere Diskettenzugriffe wie relative und sequentielle Dateien werden um Faktoren von durchschnittlich 2 bis 6 beschleunigt, je nach Anwendung. Die Kommandos »Scratch« und »Validate« sind nur unwesentlich schneller als bei der 1541. Bei der 1571 kann ein Validate allerdings doppelt so lange dauern, weil ja die doppelte Datenmenge (1328 statt 664 Blöcke) bearbeitet werden muß. Das Formatieren schließlich ist auch recht schnell: Die 1570 braucht ca. 23 Sekunden, die 1571 die doppelte Zeit, weil sie ja auch doppelseitig formatieren muß.
Während die 1570 eine leicht verbesserte Version der 1541-Mechanik verwendet, ist die der 1571 völlig neu entwickelt worden. Die Verbesserungen in der 1570 beschränken sich auf zwei Lichtschranken, die auch in der 1571 vorhanden sind. Der Hauptunterschied ist der doppelte Schreib-/Lese-Kopf der 1571, mit dem auf beide Seiten der eingelegten Diskette zugegriffen wird.
Für den CP/M-Betrieb ist die 1570/71 in der Lage, MFM-Formate (das ist ein CP/M-übliches Format) einzulesen, was aber mittlerweile nur zweitrangige Bedeutung hat, da die Standardsoftware (WordStar, dBase II, Multiplan, TurboPascal, CBasic, MS-Basic usw.) im Commodore-1541-Format lieferbar sind, das von der 1570/71 ohnehin gelesen werden kann (siehe auch 7.5.1).

Tips und Tricks zur Floppy 1570/1571

Zusätzlich zu den bekannten DOS-Befehlen gibt es bei der 1570/71 noch das U0-Kommando, das viele nützlichen Funktionen beinhaltet.

Umschalten zwischen 1570/71- und 1541-Betriebsart

Dieser Befehl dient dazu, die 1570/71 entweder in den 1541-Modus oder in den schnelleren 1570/71-Modus zu schalten. Im 1541-Modus läuft die Floppy nur mehr mit einer Taktfrequenz von 1 MHz (sonst 2 MHz) und simuliert eine 1541 sehr originalgetreu.
Die Syntax ist folgende:

```
OPEN 1,8,15,"UO>MO"     schaltet in den 1541-Betrieb
OPEN 1,8,15,"UO>M1"     schaltet in den 1570/71-Betrieb
```

Den ersten Befehl sollte man vor einen GO64-Befehl setzen, da sonst die 1570/71 beim softwaremäßigen Umschalten in den C64-Modus nicht auf den 1541-Betrieb umgestellt wird, wie dies beim Drücken von <CBM> bei einem Reset erfolgt.

Wenn im C64-Modus die 1570/71-Betriebsart verwendet wird, kommt man in den Genuß des schnellen Formatierens und der doppelten Diskettenkapazität, nicht aber des schnellen Ladens, das nur im C128- bzw. CP/M-Modus möglich ist.

Einstellen eines Schreib-/Lese-Kopfes

Dies ist nur mit der 1571 möglich und erlaubt dort das Umschalten zwischen beiden Diskettenseiten, wenn sich die Floppy im 1541-Modus befindet. Die Syntax lautet dann

`OPEN 1,8,15,"UO>HO"`	für den Kopf auf der Diskettenseite 0
`OPEN 1,8,15,"UO>H1"`	für den Kopf auf der Diskettenseite 1

Achtung! Eine Diskette, die auf einer 1541 durch Umdrehen auf beiden Seiten beschrieben wurde, kann dadurch nicht vollständig gelesen werden. Das Umschalten nützt also nur etwas, wenn die Diskette generell auf der 1571 im 1541-Modus beidseitig mit Hilfe dieses Kommandos beschrieben wurde.

Einstellen der Gerätenummer

Mit den Befehlen

```
OPEN 1,8,15,"UO>"+CHR$(x):CLOSE 1
```

stellt man die Gerätenummer »x« ein. Nehmen wir an, »x« sei 9 gewesen, so kommt man mit

```
OPEN 1,9,15,"UO>"+CHR$(8):CLOSE 1
```

in die Normaleinstellung (Geräteadresse 8) zurück.
Die Gerätenummer »x« darf zwar Werte von 4 bis 30 annehmen, aber nur Werte von 8 bis 15 können von den Diskettenbefehlen des Basic 7.0 sinnvoll verarbeitet werden. Um beispielsweise das Directory der Floppy mit der Geräteadresse 9 zu erhalten, schreibt man:

```
DIRECTORY U9
```

Es sei noch erwähnt, daß die Geräteadresse der 1570/71 auch mit den DIP-Schaltern (siehe Handbuch) eingestellt werden kann, allerdings hat die genannte Software-Lösung auch Vorteile (man muß bei der 1570 nicht erst das Gehäuse aufschrauben); außerdem kommt man kaum an die DIP-Schalter der im 128D integrierten Floppy 1571.
Im Gegensatz zur DIP-Einstellung wird die Anwahl über das Diskettenkommando jedoch bei einem Ausschalten von Computer oder Floppy bzw. einem Reset nicht beibehalten.

Die doppelseitige Nutzung von Disketten mit der 1571

Die Floppy 1571 nutzt Disketten doppelseitig, d.h. man muß nicht umdrehen, sondern behandelt bei einer eingelegten Diskette beide Seiten zusammen (je 664 Blöcke) als eine einzige Seite mit 1328 Blöcken, also ca. 340K Speicherkapazität.
Dies geht so weit, daß sogar ein Programm teilweise auf der Vorder-, teilweise auf der Rückseite stehen kann. Das Directory darf jedoch nach wie vor nur 144 Einträge umfassen, wie man es von der 1541 kennt; die Spur 18 der zweiten Seite wird nämlich nicht als Fortsetzung des Directory-

Eintrags verwendet, allerdings hat Block 18,0 auf der zweiten Seite die gleiche Funktion - BAM = Block Availability Map - wie auf der ersten.
Die Spuren 1-35 liegen also auf der Vorderseite, die Spuren der Rückseite werden mit 36-70 numeriert, da beide Seiten als eine einzige nach außen hin ausgegeben werden.

7.4 Die Kompatibilität

Da 1541 und 1570 die Disketten einseitig beschreiben und sich mechanisch nur geringfügig unterscheiden (die »neue«, Mitte 1986 erschienene 1571 ist sogar fast identisch mit der 1570, da sie auch eine Lichtschranke und einen Knebelverschluß hat), haben sie dasselbe Aufzeichnungsformat, weshalb wir dieses als 1541/70-Format zusammenfassen wollen.
Während die 1571 auch 1541/1570-Disketten einlesen kann (vor allem im 1541-Modus), ist es mit der 1541/70 nicht möglich, doppelseitige 1571-Disketten zu verarbeiten.
Die 1571 liest aber lieber eigene Disketten, da sie dann schneller ist (siehe 7.3) und ihre ganzen Fähigkeiten entfalten kann. Ihr Mißfallen drückt sie dadurch aus, daß das Initialisieren mit einer 1541/70-Diskette erst zu einem »Rattern« führt, wie man es von Schreib-/Lese-Fehlern kennt. Allerdings wird die Diskette dann ordnungsgemäß verarbeitet.
Bei Umdrehen doppelseitig formatierten 1541/70-Disketten wird nur ein leichtes Flackern der LED ausgelöst.
Ein Problem für sich ist die Verwendbarkeit von Programmen, die speziell für die 1541 entwickelt wurden. Selbst wenn diese im 1541-Modus ablaufen, stürzen sie oft ab, da das DOS von 1571 und 1541 nicht identisch ist. Oft führt nur ein kleiner Unterschied zur Inkompatibilität, was z.B. das Kopierprogramm »QuickCopy« auf der 1570/71 zum Systemabsturz bringt. Manche Programme wie »Turbo-Nibbler« und »Hypra-Load« funktionieren aber weiterhin im C64-Modus.
Oft funktionieren Programme nur im 1541-Modus, weshalb man dann nicht GO64 eingeben sollte, sondern vielmehr ein Reset bei gedrückter <CBM>-Taste auszulösen ist.
Auch mit den meisten Kopierschutzmechanismen, die sehr unübliche Befehlsfolgen verwenden, gibt es Probleme, da diese auf der 1570/71 nicht mehr funktionieren. Allerdings werden solche Programme nach und nach angepaßt.

7.5 Die drei Floppies im Vergleich

In diesem Unterkapitel finden Sie Hinweise, wie man Probleme mit den drei Floppies bewältigen kann, die aufgrund der eingeschränkten Kompatibilität entstehen. Profi-Informationen finden Sie in den Büchern »Die Floppy 1541« beziehungsweise »Die Floppy 1571«, welche im Markt-und-Technik-Verlag erschienen sind und außer einer Einführung in die Programmierung der Diskettenlaufwerke für Anfänger, Fortgeschrittene und Profis auch ein komplettes, kommentiertes DOS-Listing enthalten.

7.5.1 Software und Diskettenformate

Wenn Software im 1541-Format ausgeliefert wird, so wird dadurch zwar die doppelte Kapazität der 1571 nicht genutzt, aber die Diskette kann von allen drei Laufwerken - 1541, 1570 und 1571 - geladen werden. Wenn die Möglichkeit besteht, das Programm umzukopieren, sollte man es auf eine 1571-Diskette umkopieren, um das lästige »Rattern« zu umgehen.
Wer eine 1541/1570 besitzt, muß nicht in Kauf nehmen, daß Software nur im 1571-Format erhältlich ist, da alle momentan erhältlichen Programme auf 1541-Disketten vertrieben werden.
Schwierigkeiten kann es mit C64-Software für die 1541 geben; diese ist zwar im C64-Modus fast immer lauffähig, sofern die angeschlossene Floppy eine 1541 ist, aber mit einer 1570/71 treten wiederholt Probleme auf. Deshalb muß man fast zugeben, daß trotz der Absicht von Commodore, die 1571 zu etablieren, die 1541 das geeignetere Laufwerk ist, wenn man aktiv C64-Software weiterbenutzt.
Die gängigen Dateiverwaltungsprogramme sind aber in der Lage, von der erhöhten Diskettenkapazität der 1571 ohne Einschränkungen Gebrauch zu machen, weshalb die 1571 für ernsthafte Anwender dennoch vorteilhafter ist.

7.5.2 Softwareschutz funktionsfähig?

Bei C128-Software ist es, wenn sie im 1541-Format ausgeliefert wird, keine Frage, daß sie auch auf der 1570/71 verwendbar ist. Bei C64-Programmen, die im C64-Modus ablaufen sollen, kann es mit der 1541 kaum Probleme geben, und wenn dies der Fall ist, so liegt es am Computer (wie Kapitel 5 zeigt, ist der C64-Modus nicht 100%ig kompatibel) und nicht an der Floppy 1541. Anders ist es bei C64-Software, die mit einer 1570/71 betrieben wird; kaum ein Kopierschutzmechanismus funktioniert weiterhin, sofern er nicht allzu »billig« ist. Da erst ein fehlerfreies Funktionieren der Softwarschutzmechanismen ein einwandfreies Ablaufen des Programms selbst ermöglicht, sollte man sich im klaren darüber sein, daß C64-Software auf Diskette in den meisten Fällen nur mit der Konfiguration C128/Floppy 1541 arbeitet, wenn sie ungeschützt ist. Da es im Grunde kein gutes C64-Programm ohne Kopierschutz gibt, sollte man vor der Anschaffung des Programms oder der Floppy 1570/71 einen Test durchführen, was leicht möglich ist, da in fast jedem Geschäft, das C64-Software führt, auch ein C128 mit einer Floppy 1570/71 steht. Am besten ist es aber, wenn man seine 1541 beim Umsteigen auf den C128 nicht verkauft, sondern für solche Programme, die mit der 1570/71 nicht kooperieren, aufhebt und bei Bedarf an den C128 anstelle der 1570/71 anschließt. Im nächsten Abschnitt erfahren Sie dann, wie man Kompatibilitätsprobleme dieser Art mit einem 128D beseitigt.

7.5.3 Anschließen des VC 1541 an den 128D

Da der 128D ein eingebautes Diskettenlaufwerk (Typ 1571) hat, daß nicht einfach ausgebaut werden kann, um ein anderes (beispielsweise die 1541) anzuschließen, sei hier für die vielen 128D-ler eine Hilfslösung zusammengestellt, an die Sie sich bitte genau halten (beim C128 kann die Floppy ohne weiteres angeschlossen werden, sofern der C128 ausgeschaltet ist):

1) Alle Geräte ausschalten!
2) 1541 am seriellen Bus anschließen
3) Nur den 128D einschalten, nicht die 1541. Dabei ist die <CBM>-Taste gedrückt zu halten (C64-Modus-Anwahl).
4) Folgenden Befehl im C64-Modus eingeben: `OPEN 1,8,15,"UO>"+CHR$(9):CLOSE 1`
5) 1541 als letztes Gerät anschalten

Jetzt wird die 1571 unter der Geräteadresse 9 angesprochen werden, während die »normale« Adresse 8 die 1541 anspricht. Statt 9 kann man der 1571 des 128D auch andere Adressen geben. Mit der 1541 ist im C64-Modus wesentlich mehr Diskettensoftware lauffähig (fast 100%) als mit einer 1570/71. Das Anschließen der alten Floppy löst somit nahezu alle Kompatibilitätsprobleme.

Stichwortverzeichnis

